HDRI 3D

LIGHTING THE WAY FOR DIGITAL ARTISTS

ISSUE #5

CaféFX and *Sin City*

INSIGHTS & TECHNIQUES from Professional Digital Artists

CAFEFX

Issue 5 $10.95 USD $13.50 Canada

SUBSCRIBE TO

HDRI 3D

MAGAZINE

Lighting the way for digital artists, HDRI 3D presents real world insights and emerging trends in digital production from the digital artist's perspective. Concepts and techniques are shared through production case studies and software specific tutorials by professionals in film production, architecture, advertising, medicine, and other fields. Topics are wide-ranging, covering areas such as Workflow, Visual Effects, Compositing, Lighting, Rendering, Modeling, Scripting, Animation, Character Rigging, Texturing, and more.

Go to www.hdri3d.com for more info!

ONLINE SPECIAL OFFER

☐ US $49 ☐ Rest of World $68

Subscribe to HDRI 3D Magazine
online before August 30, 2005
to get this great discount!

Six issues a year

4 EASY WAYS TO ORDER

1. In our online store at www.dmgpublishing.com
2. Fax to 1-702-616-9647
3. Phone 1-888-778-9283 or 1-702-990-8656
4. Mail to:

DMG Publishing
2756 N. Green Valley Pkwy #261
Henderson, NV 89014
USA

Name ______________________
Address ______________________
____
City/State ______________________
Zip/Postal Code ______________________
Country ______________________
Telephone ______________________
Email ______________________

PAYMENT

☐ I'm enclosing a check in US Dollars payable to DMG Publishing
☐ I wish to pay by credit/debit card

Card# ______________________ Exp Date __________
Signature ______________________ Date __________

Lighting the way for digital artists

From the Editor

From the Editor
From the Editor

Dariush Derakhshani
Editor-in-Chief

It would be sexy to say that Dan Ablan and I were involved in a steel cage death match recently, or that I uncovered a vast conspiracy that toppled Dan's iron tight grip on HDRI 3D magazine to wrest control and seize the vast empire that lay at his feet. Quite the opposite. Dan has been a steady navigator for this magazine, and led it to a wonderful position that we are all proud of. But along with the trappings of being good at what you do come the time constraints that tear at you from all directions, and we are all sorry as well as proud to see Dan moving on to focus on his thriving businesses. Though the steel cage death match is still a nifty idea.

I am honored to pick up the reigns from him and to carry on the torch, to mix my metaphors and sound like a dork. And while it is true I have a heavily Maya background, I strongly feel the business of making CG is cross platform and software independent. The mantle of making CG is on the artist, not the software, a point that is too often glossed over and forgotten like an anniversary. It is my intention to see HDRI 3D magazine explore the CG world from as much an agnostic stance as possible, and in doing so, we will just happen to cover as much of what's out there as possible, from LightWave to Maya, from After Effects to Shake, from mental ray to Renderman.

But as Bach said, if it ain't Baroque, don't fix it. Just like the steel cage death match idea, it would be sexy as hell to run in here with a bullhorn in one hand and a bamboo switch in the other, screaming for everything to change. I want my face featured on every page! I want the font to fit the curvature of my toes! I want all the writers to type in the nude (although some already do)! The fact of the matter is that HDRI 3D magazine is gaining popularity and a more solid reputation among artists, teachers, and students as being a good resource for information, and that is certainly due to the standards already established with Dan at the helm. So it would be foolish to make everything different. But I do have some very definite goals in mind (with only a couple of them involving nudity and a steel cage):

1. **Listen closely.** This industry is like no other. It is strongly based in the will, wants, and needs of its community. Successful software makers listen to what their users want, and so shall we. But this also means we need to hear it from you. It is important to all of us to hear from you all the time. Log onto our site and rate our articles, send us emails with thoughts and needs; let us know what you think and tell us what you'd like to see.

2. **Attract top talents.** It's important to pull top talent to the magazine's pool. And I don't mean just for writers, but for readers as well. The more educated our readers, the more educated the magazine. Finding really good tutorial content for CG is tough. There is a real draw for people to go to the Internet, since it holds a veritable trove of free stuff. What we have begun and will continue to do is create tutorial content that is checked for accuracy, and edited and developed by several working professionals to make the content as tight and topical as possible; so much so that it surpasses anything you would find free online. And for this, we enlist some of the best pros out there to share their workflows and ideas with you. This is a tough thing; no one wants to blithely give away what makes them good and marketable. But we're getting them using our vast powers of mind-control, bending them to our will. Muahahahahahahaaa.

3. **Go behind the scenes.** Knowing the basics of how a studio finished a project is really important to understanding how to do things yourself, whatever your position in the industry, hobby, or academia. Productions like commercial effects, feature films, and music videos are living, breathing creatures involving a great depth of intricacy; you'll always learn something by studying how *any* production is completed. By mixing production stories that cover pipelines and workflows with pointed software tutorials and concept articles, we hope to round out the experience for our readers. Keep your eyes peeled.

OK, so enough of that BS, let's get to the meat of Issue #05. I am stoked about the continuation of our mental ray series. Here, Boaz covers all you ever wanted to know about lighting and rendering different passes that all come together in our next issue to composite the perfect frame, and it's appeal really crosses to any 3d renderer. Our ever popular friend Deuce has some great insights on how to create particle based bullet hits in LightWave, to save the agony of having to place powder squibs everywhere on set. Robin gives us the beginnings of finding your Zen with MEL and understanding how programming with it works in the first of a new MEL series. Peter shares the third part in his biped rigging opus, and Matt makes some SOFTIMAGE|XSI faces at us. And of course, we catch Dan trying to tamper with Mother Nature when he splits cells in an advanced Lightwave tutorial. Naughty Dan...

Whatever your flavor of software happens to be, check out all the pieces. What you learn about one package will most definitely help you in the others. Hell, even if you work off the box entirely, it's good to understand the techniques to make managing a production easier, if not make you sound smarter.

IN THIS ISSUE

departments

3 FROM THE EDITOR
BY DARIUSH DERAKHSHANI

6 FAREWELL
BY DAN ABLAN

6 JOB PERSPECTIVES FOR SOFTIMAGE|XSI ARTISTS
BY RAFFAEL DICKREUTER

7 STUFF
BY BRAD CARVEY

80 LIGHTWAVE ESSENTIALS: LSCRIPT COMMANDER
BY BRAD CARVEY

82 FINAL RENDER: POOPING IN THE RING
BY ANDREA CARVEY

feature story

8 THE MAKING OF SIN CITY
A behind-the-scenes look at how CaféFX helped create the gritty world of *Sin City*.
BY CHRISTINE BUNISH

tutorials

14 FINAL PREPARATIONS PRIOR TO BINDING THE MODEL
Part 3 of Peter Ratner's *Setting Up the Human Model for Animation* series.
BY PETER RATNER

22 SPLITTING CELLS
Often applying image maps to HyperVoxels leads to unwanted results when animated. But by using LightWave's integrated HyperVoxel textures, you can achieve excellent results.
BY DAN ABLAN

26 MENTAL RAY FOR MAYA PART 2
Boaz explains how to render custom passes with mental ray and explains how to control the Framebuffer.
BY BOAZ LIVNY

35 I FEEL LIKE I'M GOING INTO ANOTHER DIMENSION...
With theater releases of *Steamboy*, *Appleseed,* and *Innocence* here in the States, it's no secret that Anime has finally reached the attention of your average moviegoer. Here are 20 quick tips that should help get you on the path to making your own CG Anime.
BY WILLIAM "PROTON" VAUGHAN

40 UNCLE ROBIN'S MINI GUIDE TO MEL, VOLUME 1: THE WAY OF PROGRAMMING
An introduction to programming with MEL in a way that makes it easy to learn, use, and debug.
BY ROBIN SCHER

47 PARTICLES AND HYPERVOXELS—THEY GO HAND IN HAND
Particles and collisions for bullet hits and the new gradient for HyperVoxels: Relative Particle Age.
BY DEUCE BENNETT

50 COMPOSITING 101: LAYERS IN DIGITAL FUSION
Explore the flexibility we have when working with layers produced from any favorite 3D application in order to deliver a better final version of our creative vision.
BY GREGORY GLEZAKOS

56 MODELING A CREATURE WITH THE EXTENDER TOOL
By the time you're finished with this lesson, you will have mastered LightWave's Extender tool.
BY ROBERT CALANDRIELLO

Resource files and selected screen shots for this issue are posted at **www.hdri3d.com/resources**

PAGE 50

PAGE 75

contents

tutorials

60 MAKING FACES IN SOFTIMAGE|XSI
Creating blendshapes and controllers within SOFTIMAGE|XSI using the advances made with v.4.0 to create an economical and fast setup. Learn to create 3D control sliders to combine different shapes and the expressions needed to link these to the shapes.
BY MATT MORRIS

64 FACIAL EXPRESSIONS: USING ZBRUSH AND MAYA TOGETHER
Pixologic's ZBrush is starting to make a huge impact in the CG animation and modeling industry. This tutorial shows one of the many ways in which incorporating ZBrush into your Maya animation workflow can make an arduous task, such as creating blendshape targets for facial animation, easy and fast.
BY ERIC S. KELLER

72 INTRODUCTION TO MENTAL RAY PRORAMMING
An overview of how even those with only limited coding skills can leverage some of the power of mental ray to create basic tools to enhance their pipeline, using the example of a reflection shader with distance-based falloff.
BY ALAN JONES

75 INTRO TO NORMAL MAPPING
Creating normal maps within a production pipeline for next-generation game engines, a technology fast becoming a prerequisite skill for all artists working on projects for new game engines and platforms.
BY JAKE CARVEY

online xtras

CHARACTER ANIMATION AND RIGGING FOR LIGHTWAVE
This article describes how to use Maestro to auto-rig and then animate your character.
BY BRIAN PACE

Access Online Xtras at:www.hdri3d.com/xtra

HDRI 3D
Issue #5

PUBLISHER
Alice Edgin alice@hdri3d.com
EDITORIAL DIRECTOR
Charles Edgin charles@hdri3d.com
EDITOR-IN-CHIEF
Dariush Derakhshani dariush@hdri3d.com
EDITOR
Brad Carvey bradcarvey@hotmail.com
EDITOR
Raffael Dickreuter raffael@hdri3d.com
TECHNICAL EDITOR
Philip Duncan philipd@hdri3d.com
MANAGING EDITOR
Amber Goddard amberg@hdri3d.com
ART DIRECTOR
David S. Wilkinson david@sabredesign.net
LAYOUT/DESIGN
Sabre Design artists@sabredesign.net

CONTRIBUTORS

Dan Ablan	*danablan@agadigital.com*
Deuce Bennett	*deuce@cimagineering.com*
Christine Bunish	*cbpen@aol.com*
Robert Calandriello	*rcalandr@optonline.net*
Andrea Carvey	*andreacarvey@hotmail.com*
Brad Carvey	*bradcarvey@hotmail.com*
Jake Carvey	*jake@spinoffstudios.com*
Gregory Glezakos	*gregory@sub-pixel.com*
Alan Jones	*alan@xsibase.com*
Eric S. Keller	*kellerrific@yahoo.com*
Boaz Livny	*info@visionanimations.com*
Matt Morris	*matt@mattmos.com*
Brian Pace	*support@stillwaterpictures.com*
Peter Ratner	*ratnerpj@jmu.edu*
Robin Scher	*robin@uberware.net*
William "Proton" Vaughan	*Proton@spinquad.com*

HDRI 3D MAGAZINE ADVISORY BOARD
Dan Ablan • Timothy Albee • Akiko Ashley
Jack "Deuce" Bennett II • Stephen M. Burns
Kevin Cahill • Dwayne J. Ferguson
Todd Grimes • Ed Harriss • Varto Keshishian
Policarpo • William Vaughan

ADVERTISING SALES 1-702-990-8656
advdmg@dmgpublishing.com

SUBSCRIPTIONS & BACK ISSUES
Online Store www.dmgpublishing.com
Email mag@hdri3d.com
Phone 1-702-990-8656 or 1-888-778-9283

CHANGE OF ADDRESS moving@hdri3d.com

HDRI 3D MAGAZINE
(ISSN 1551-689X)
is published bimonthly by DMG Publishing
2756 North Green Valley Pkwy #261
Henderson NV 89014 USA

© 2005 by DMG Publishing. All Rights Reserved. No part of this publication can be reproduced or transmitted in any form or by any means, electronic or mechanical including photocopying, recording, or information storage and retrieval system without permission in writing from the publisher.

Dan Ablan
Outgoing Editor-in-Chief

Farewell

I often wonder where the time goes. As a child, I was always waiting for something, such as school to end, or Christmas vacation, or that next birthday. I remember the adults saying not to rush it, to "enjoy" my time. But did I listen?

Now, some 30-plus years later, as 40 rears its ugly head, I reminisce about the carefree childhood days. What is it that makes things so different from childhood to adulthood? Is is the fact that we are so aware of the world around us that we come bitter? Perhaps it's the daily pressure of work and bills? Or maybe it's simpler than that. Maybe the happiness we have as a child can be achieved as an adult. When you think about it, being happy is often nothing more than having something to look forward to.

For me, looking forward means finding additional challenges in my professional life. Fortunately, my company, AGA Digital Studios, Inc., has had a record year in the 11 years we've been in business, and this year is no different. And our subsidiary company, 3D Garage.com, continues to grow every month. The demand for our training materials is overwhelming. Therefore, this will be my last editorial for HDRI 3D as I step down as Editor-in-Chief to focus my attention solely on my business, and my new publishing and training ventures.

I'd like to thank Alice and Charles Edgin at DMG Publishing for the opportunity to edit both Keyframe and HDRI 3D Magazines these past two and a half years. I'll miss working with Amber Goddard, who truly is the glue that holds this magazine together. My regards go to Brad and Andrea Carvey for their great technical editing and "Final Render" each month. HDRI 3D's contributors, as well as David Wilkinson's excellent layouts, have made this magazine shine each issue. Keep up the great work!

Finally to the readers, please keep me posted on your ongoing learning. I'll be popping in from time to time with articles on LightWave, modo, and some other cool products coming down the pipe. Be sure to keep an eye on 3D Garage.com for out new training products and as always, keep on learning.

Raffael Dickreuter
Editor

Job Perspectives for SOFTIMAGE|XSI Artists

I get emails from people that have used other packages and ask if it would be worth it to invest a lot of time to learn SOFTIMAGE|XSI well. They are insecure about how their job perspectives look for the future.

I often hear that new people like the fast workflow and the slick interface a lot and they would love to spend more time with it, but if the job perspectives don't seem that promising, people have doubts. It seems the situation has improved over the years, as more and more studios are adopting SOFTIMAGE|XSI as their main 3D tool or use it as part of their pipeline. A good example is London-based game company Lionhead Studios, which recently switched its pipeline to SOFTIMAGE|XSI. Still, it has to be said that most large studios use either Maya, 3ds Max, or proprietary software. It's clear that a lot of people out there with a lot of experience are 3ds Max or Maya savvy; this also obviously means way more competition for each job you apply for. So, at the moment, there might be fewer jobs out there where SOFTIMAGE|XSI knowledge is required, but if you are looking for a SOFTIMAGE|XSI job, you will not face the same amount of competition as 3ds Max or Maya users.

Good SOFTIMAGE|XSIartists are hard to find and are therefore in high demand. A good example was the last Los Angeles SOFTIMAGE|XSI user group, where several studios were desperately trying to fill their open positions. However, a common problem seems to be that many people that would like to use the software and know it pretty well live in a city where it has little market share, whereas they don't stay in areas where they would be in high demand, bummer. Basically, it means SOFTIMAGE|XSI jobs don't pop up just everywhere.

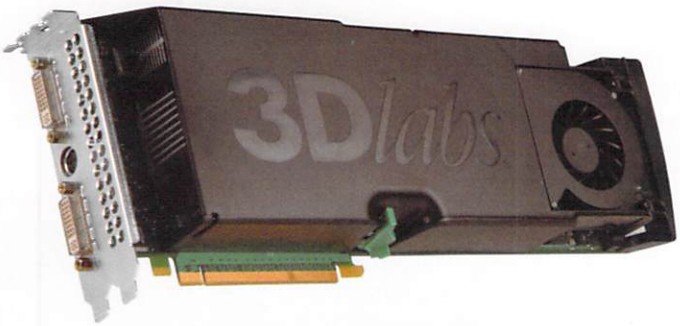
The big question is how the situation will change in the upcoming years. It's hard to predict, but considering a few aspects, the future may be brighter than it seems. SOFTIMAGE|XSI is the most modern software out there now; obviously it can take advantage of not making the same mistakes other developers did and having a pretty open architecture, even allowing many different scripting languages, which is pretty unique. Softimage did its rewrite when moving on from SOFTIMAGE|3D; NewTek, Autodesk, and Alias might have to consider doing this sooner or later, too. Maybe then, the time comes when many more studios want to integrate SOFTIMAGE|XSI, and artists who know the software well, and have a great reel, will be even in more demand than now. Let's see what new features Softimage will unveil at SIGGRAPH, and let's hope it has an impact.

STUFF

By Brad Carvey
Electrical Engineer, 3D Animator
New Mexico

Last year, I reviewed a $600 OpenGL graphics card. It was a new card that worked great on one monitor, but LightWave had problems with displaying OpenGL graphics on the secondary monitor. I have had problems with other highend graphics cards in the past. I still use a three-year-old Fire GL 4 that has problems playing my Windows media files. It seems like, in the past, very expensive graphics cards designed for CAD applications sometimes had problems with my graphics applications. For example, I have had problems with the Video Toaster displaying video, on the VGA monitor, when used with a $1600 graphics card, but it works fine with more mainstream cards that cost a few hundred dollars.

I have been reviewing a **Wildcat Realizm 800 graphics** card for several months. At a MRSP of $2,799, it's the most expensive graphics card I have ever used. The good news is that it's worth it. The Wildcat Realizm 800 graphics card has been perfect. I have not had any problems with the software or hardware, even though I received a pre-release version of both. Based on my testing and the testing results that I have seen in other reviews, the 800 is the fastest OpenGL board available. If you are a professional and work with complex objects in a 3D program like Maya or LightWave, then you should try the 800. After all, time is money.



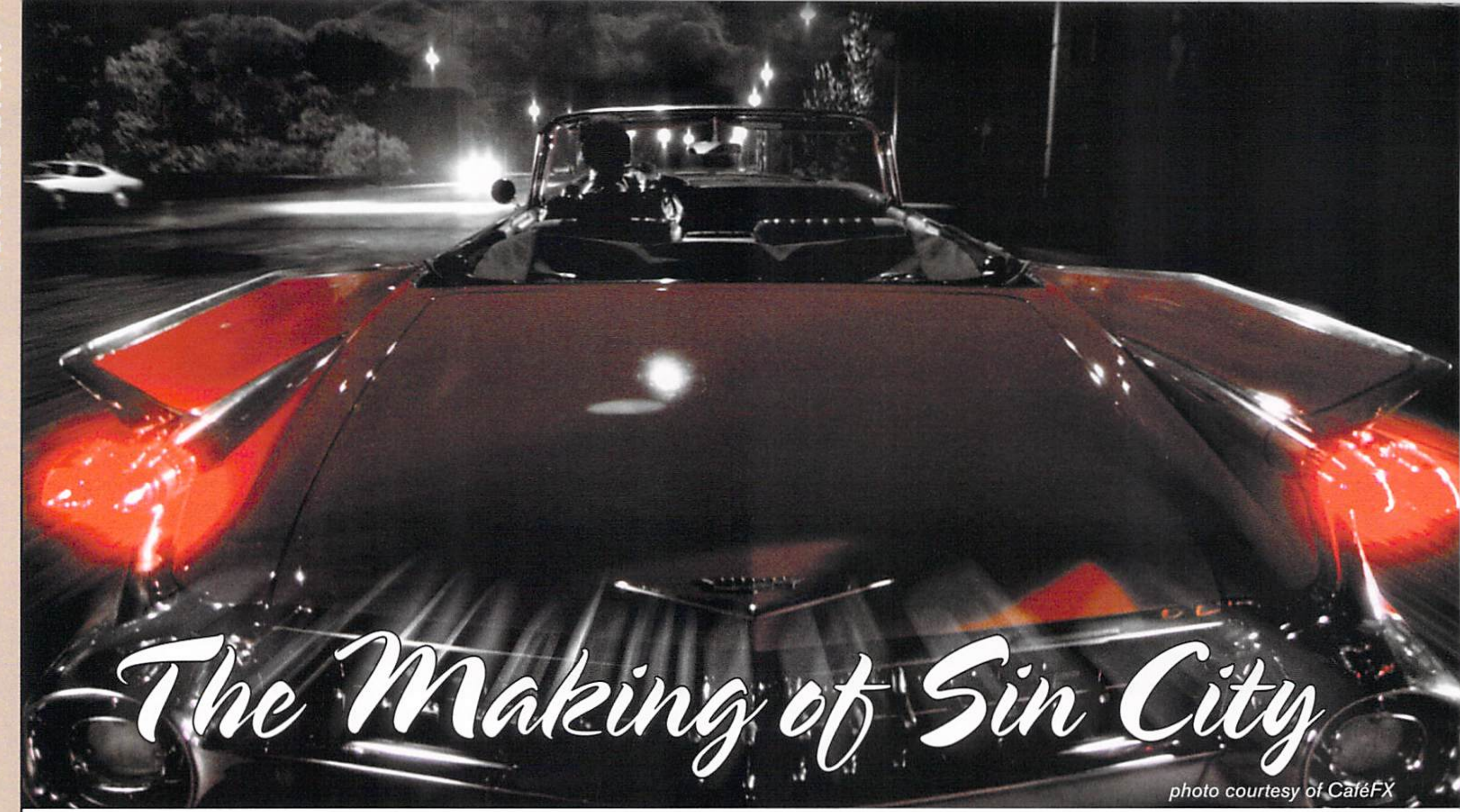
To test the OpenGL performance of graphics cards, a free program called **SPECviewperf** ***www.spec.org*** is typically used. Basically, the SPECviewperf program rotates a complex object as fast as possible. An inexpensive graphics card might rotate the object once a second. The expensive cards might rotate the same object hundreds of times a second. The faster the card can redraw the OpenGL object, the more revolutions it will be able to complete. The SPECviewperf program also has the ability to test application-specific viewsets. The viewsets make it possible to test the relative OpenGL performance of different applications on different graphics cards. The SPECviewperf program has a Maya viewset, but not one for LightWave.

OpenGL is a computer graphics programming language compiled and then executed by the graphics processors on your graphics card. Some graphics cards are faster at executing the OpenGL instruction than others, but it's also important to realize that applications implement their OpenGL support differently. I tested the 800 on a 3.2 GHz Gateway Gr500 computer. The SPECviewperf Maya result on my computer was approximately 51. I have seen results as high as 65 on faster dual-processor computers. The application, graphics card, and the computer all affect the OpenGL speed.

NewTek recently hired Jay Roth and Mark Granger. Jay Roth is now the President of NewTek's 3D Product division. This is very good news. Jay and Mark were co-founders of a rival 3D animation company, called Electric Image. Together they probably have around 40 years of 3D graphics experience. I expect to see LightWave improvements and innovations, including better OpenGL performance.

A few years ago, I bought two 17" LCD monitors with a $1600 graphics card. The IBM monitors were about $1100 each. Recently, a friend of mine bought ***three Sony 19" LCD monitors with a custom three-monitor stand for less than $1500.*** The monitors bolt on to the stand and adjust to fit together perfectly. The stand sits on the desktop, but wall mount versions are also available. I was really impressed with the quality of the monitors and the stand. The Ergotron's Triple SIDE-BY-SIDE Flat is available for between $232 and $299. The Sony monitors were Sony SDM-S93s and cost less than $400 each.

Brad and Andrea Carvey *have been doing computer animations for a long time. in 1969 brad used an analog computer, which was the size of a car, to produce his first computer animation. Andrea, an archeologist, prefers to do scientific animations; her credits include programs like Discovery Channel's* ***Understanding Cars.*** *Brad is an electrical engineer and an emmy award-winning member of the Video Toaster development team. He prefers to do feature film work. his credits include films like* ***Men in Black, Stuart Little, Black Hawk Down, Kate & Leopold,*** *and* ***Master of Disguise.***



The Making of Sin City

photo courtesy of CaféFX

By Christine Bunish
Freelance Writer
Cedar Grove, New Jersey

It's rare for a CG and visual-effects studio to have the chance to contribute more than 600 consecutive shots to a feature film, so when ComputerCafe Group's CaféFX was asked to create 45 minutes of CG backgrounds and effects for "The Big Fat Kill" episode in Sin City, it was both a luxury and a challenge to follow the dark narrative from start to finish.

"Being able to take charge of an entire mini-film was very rewarding," says VFX Supervisor Jeff Goldman. "Usually, as a visual-effects vendor, we work on particular sequences within a film, but this was the first time we were given an opportunity to work on an entire film, albeit a shorter one, in which every shot was an effect.

"Unlike most other films where visual-effects integration provides the challenge and the look of the original film plate dictates the consistency of the look of the visual effects, *Sin City* was an exercise in integrating the live action into the visual effects, which was something of a reversal."

Directors Robert Rodriguez and Frank Miller shot the live-action elements of *Sin City*, based on Miller's hard-boiled graphic novels set in the fictional metropolis, on greenscreen at Rodriguez's Troublemaker Studios in Austin, Texas. CaféFX got the charge to devise black-and-white environments for "The Big Fat Kill," a nighttime tale brimming with booze, broads, and bullets starring Clive Owen and Benicio del Toro.

CaféFX invested in 35 additional Boxx Technologies dual AMD Opteron-based workstations to accommodate the eight-month project, which was five times as big as anything they'd done to date. The company also added an Apple Final Cut Pro editing system with Blackmagic Designs DeckLink HD Pro 4:4:4 capture card and Sony's SRW-5000 HD deck. Render power came in the form of a Verari Systems BladeRack server incorporating 50 dual-processor Xeon blade units in addition to 70 Boxx render rack units based on dual AMD Opteron CPUs. Although "The Big Fat Kill" ultimately consumed 5-6 TB of data, storage upgrades included a 7 TB group of Isilon Systems clusters and a 6 TB AMD-based Teran storage server from fsix servers.

CaféFX also bolstered its talent infrastructure using custom software and a database of over 5,000 freelancers to staff up for the project.

SIN CITY FROM PAGE TO SCREEN

CaféFX was no stranger to crafting CG and visual effects for greenscreen sequences having created shots for Rodriguez's *Spy Kids 3-D: Game Over* and Kerry Conran's ground-breaking *Sky Captain and the World of Tomorrow*.

photo courtesy of CaféFX

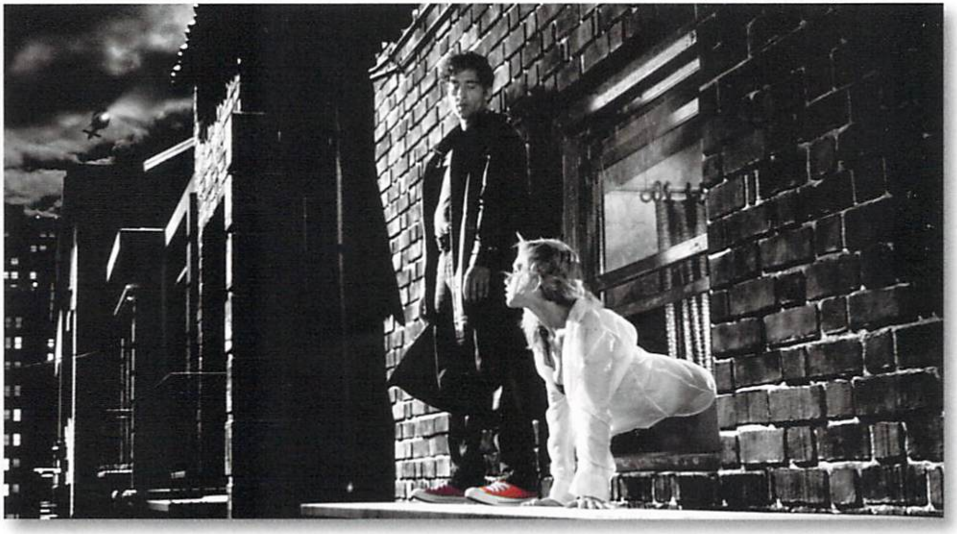


photo courtesy of CaféFX

For "The Big Fat Kill" episode of *Sin City*, it was the company's goal to recreate Miller's graphic novel on the big screen, filling Sin City's Old Town with a maze of realistic, grimy alleyways and back streets that ultimately lead to a rural tar pit: the ideal place to conceal a crime.

"Rodriguez shot very true to the original book, down to very similar camera angles and compositions," says Goldman. "Creatively, he gave us quite a bit of leeway in realizing *Sin City*. He trusts his vendors to give him something good. He concentrated on filming the characters and the action, but we were very responsible for the look and content of the film. The environment we built for *Sin City* was developed by us."

CaféFX used NewTek's LightWave 3D almost exclusively for the alleys' modeling, texturing, rendering, and lighting. Its primary compositing tool was eyeon's Digital Fusion.

"We tend to rely on LightWave 3D a great deal due to our comfort level with the package, as well as its ease of use for the layout work that dominated *Sin City*," says Goldman. "It's a very quick and efficient package for the type of work we were doing and makes it very easy for a single artist to take a shot from the model stage to the rendering stage. But LightWave is just one tool in our ever-expanding arsenal. We also used Alias Maya for character animation and the big car crash sequence in 'The Big Fat Kill.'"

CaféFX began construction of Sin City very practically with in-house Art Director Peter Lloyd taking digital still photos of exteriors and interiors of run-down LA architecture of the 1920s and 30s. The CaféFX team studied *film noir* classics, noting the look and decor of the streets and apartments; they also screened *Eraserhead* for its distinctive look and reviewed photo references of the famed La Brea tar pits.

"Sin City itself is somewhat timeless. It's a place where the old can meet the new, and the period of the film isn't specific," notes Goldman. "In the distance you see newer skyscrapers and modern buildings of downtown, but the Old Town in which our story takes place is populated by three-story brick structures that all look run down – in some cases collapsed and broken – and peppered with graffiti.

"The buildings, graffiti, fire escapes, trash cans, trash – everything you'd normally put on a set we had to build and texture with real world pictures of brick and asphalt, for example, or procedurally-generated textures."

photo courtesy of CaféFX

VENTURING INTO BACK ALLEYS

At CaféFX's Santa Maria and Santa Monica, California studios, concurrent streams of compositing and CG teams went to work.

In the compositing stream, teams used hand rotoscoping and traditional key pulls to extract actors from plates in over 600 shots. Adding complexity to the process was the 10-bit 4:4:4 color space of Sony's HDCAM SR, the new HD format used by Rodriguez for production.

"Our 3D and compositing packages handled the footage just fine, as they were always designed for high-bit depth imagery," says Goldman, "but our Apple Final Cut Pro editing system presented a series of bleeding-edge problems.

"On HD features, we normally receive HD cloned tapes of the original footage, which we digitize in-house into our editing system. At the onset, the HDCAM SR equipment was so new that there was a delay in getting the SRW-5000 HDCAM SR deck, and we had teething issues with the first iterations of the Black Magic HD capture card and drivers. So it was a good month or two before our capture system could somewhat reliably be put into production."

Since HDCAM SR was developed "with more bandwidth, which retains full-frame resolution for luminance and chrominance channels, it ultimately allowed for more accurate data for our tools to work with," Goldman points out.

The CG stream crafted 3D animatics to rough out the geometry of the alleys and map out camera placement.

"The graphic novel was a good reference for iconic frames, but the book and dialogue are essentially one-shots," notes Goldman. "Anything after the iconic shot, anything that made it more filmic, was up to us.

"Unlike a dressed set, when you're shooting greenscreen, there's no frame of reference for orientation," he explains. "You don't know where the characters are in relation to the environment or to each other. So blocking things out in 3D, rendering in LightWave, comping in Digital Fusion, and cutting them in Final Cut helped a lot."

The 3D animatics went to Rodriguez for approval, and the CG team began texturing buildings to final quality. Texture material came from reference photographs and location photos of various types of environments and mixed with procedural textures. Source lighting and dramatic set lighting were also added.

"Rodriguez used a lot of side and back lighting on the characters, which worked really well at night and for such a dark story," says Goldman. "He also shot with a static camera for the most part, so we were able to get away with single-frame renders, which was a huge timesaver."

At this point, the 3D elements went off to the compositing team for additional enhancement.

"You're not going to get 100 percent of the details right in 3D: it would take too long. So in compositing, we manipulated images further by adding and reducing lighting and reflections, treating shots much like big matte paintings," Goldman explains. "You get a very vibrant and flexible look in 2D, and, with up to 20 iterations per shot, it's much faster."

WHEN IT RAINS, IT POURS

The night setting of "The Big Fat Kill," along with its gritty locations and seamy story, are made darker still by a storm which threatens to break in the first third of the episode and a subsequent downpour throughout the rest of the episode.

"Rodriguez had shot rain elements in HD that were to be used in a sequence in one of the other *Sin City* stories, but it was far too heavy for us to use in ours," Goldman recalls. "So we referenced the rain plates and analyzed exactly how they looked, why they looked the way they did. Then we created a flow in Digital Fusion that recreated the look of the HD rain but allowed us to edit factors such as rain direction, rain spread, randomness, and density.

"Each visual drop was first created and then driven with Digital Fusion's internal particle system in a combination of 3D and 2D. Digital Fusion allowed us to have 3D access to the behavior of the rain particles, but utilized the low memory and quick processing of a 2D system. This gave us a fast way of getting layers of rain in the environments."

Goldman rendered out 1,000-frame sequences of various rain conditions, creating a library of different kinds of rain, which also saved time. CaféFX tapped its inventory of smoke, steam, and fog elements, which were layered into shots to create a more atmospheric look.

"For rain bouncing off live-action characters, we again utilized Digital Fusion to throw up little particle fountains that we integrated with the mattes of the actors," Goldman

photo courtesy of CaféFX

Mike's
LIQUOR

photo courtesy of CaféFX

photo courtesy of CaféFX

photo courtesy of CaféFX

continues. "While not actually interacting with the surfaces of the actors in 3D space, it did allow for a bit of interaction that is just enough to distract the eye and give the impression that the virtual rain was bouncing off the real actors. It's all an illusion!"

For rain dripping down the environments of Sin City, CaféFX used a combination of live-action elements shot outside its Santa Maria studio and mapped onto 3D objects as image sequences or as elements in the comp. Animators also used procedural textures.

For a handful of shots requiring the camera to travel through the environment, such as dolly or crane moves, CaféFX tracked shots in 3D and used Maya particles to drive the Digital Fusion-developed rain to give more flexible 3D results.

"While you can move Fusion's virtual camera, you can't yet import 3D data from another package," Goldman notes. "And Fusion's 2D sprite system driven with three-dimensional particles means that sprites are always flat onto the camera so you can't get a huge amount of perspective. So, we opted to use Maya for these shots. We custom-coded a particle system to replicate our results in Fusion and drove those images with Maya's particle engine."

COLORING A BLACK AND WHITE WORLD

Although Sin City's environments are stark black and white, "The Big Fat Kill" is not devoid of color. Splashes of color, determined by Rodriguez, appear throughout the episode, and a 10-minute sequence in which Dwight (Clive Owen) drives a car in the company of a corpse introduces a colorwheel effect of changing red, yellow, blue, and green light to reflect Dwight's disintegrating state of mind.

"The colorwheel effect is a more unique color application in the story," notes CG supervisor David Lombardi. "Most color effects were quite literal, but the colorwheel effect leaves the viewer wondering where the color is coming from and shifts you out of the space of normal color to a world where Dwight is losing his grip on reality."

photo courtesy of CaféFX

According to Lombardi, CaféFX delivered 170 shot modifications in color.

"Most of the cars in *Sin City* were in color for the final film, and this was just done in the CG renders. For live-action characters, color was overlaid on the black-and-white footage, mixing a percentage of the original color value of a particular pixel while retaining luminance values. Usually, because of the need to extract actors from greenscreens, the masks needed to hold out areas of color were readily available. In other situations where specific areas are held out – a character's eyes or clothing, for instance – roto-masks were employed."

Quentin Tarantino, who directed the car sequence, wanted to employ a psychedelic lighting effect reminiscent of Dario Argento's 1970's Italian horror film, *Suspira*, and he used limited colored lighting on set during the greenscreen shoot.

With Tarantino's footage in hand, CaféFX sequence lead compositor Brian Fisher "came up with a technique where he isolated existing color channels in the original photography and used them as a starting point to apply colors back onto the actor," explains Lombardi.

Roto-masks were used to define the actors' faces in terms of screen left and screen right for light sources so colors could be applied to one side of the face or another; a procedural element gave a sense of movement within the lighting effect. Tarantino scripted the color pattern to his specifications to prevent the application of two colors to the same area simultaneously: the application of one color required the reduction of the other.

Tarantino used a practical car in only a few of the shots, since his very dramatic camera angles would have been impossible to achieve within a confined space. So, many of the greenscreen plates for the sequence feature the actors sitting on green blocks with a prop steering column. As a result, Luke McDonald and Lee Carlton respectively modeled and textured a CG car in LightWave.

The sequence's storm is punctuated by lightning hits. Timing all of the existing lightning hits on the CG car was done using a "probe" in Digital Fusion to analyze certain parts of the frame and provide an envelope to control the amount that different CG-rendered "light passes" were applied to the scene. So, lightning flashing on the original greenscreen plate resulted in the addition of a CG element to the scene with the exact same timing. This made matching the complex lightning flashes much easier, Lombardi reports.

SINKING INTO TAR PIT OOZE

Dwight seeks to conceal the corpse in his car by driving out to the tar pits, which are also the site of an abandoned amusement park.

"It's the ideal spot for skullduggery," notes VFX Supervisor Everett Burrell. "We created a broken-down Ferris wheel, signs, debris, and fiberglass dinosaurs stuck in the tar."

The tar pit terrain posed a challenge different from that of Old Town's back alleys, he points out.

"It was especially difficult because it was an organic environment. We had to have grass swaying in the wind, trees, and rain dripping off the dinosaurs."

The sequence reaches a fiery climax when mercenaries toss a grenade under Dwight's car. The vehicle explodes, and Dwight and the car propel into the air, landing in the tar, where they start to sink in the viscous liquid.

Domenic DiGiorgio used Maya to craft a digital Dwight, seen in medium and wide shots during the pyrotechnics. He created the explosion in 3ds Max with After Burn for smoke and other volumetric effects; he did the gooey tar simulation in Next Limit's Real Flow.

Tar-pit lead animator Alex Friderici devised the area's organic environment and lighting scheme in LightWave; he then exported these files to Maya so DiGiorgio could add Dwight's character animation to the grim world.

Ultimately rescued from the ooze, Dwight finds himself in a final car chase and crash done completely in Maya by a team of Maya-savvy artists.

"The sequence was pre-vized by Troublemaker in SOFTIMAGE|XSI, and the basic camera and terrain were imported into Maya," says Burrell. "We referenced a lot of high-speed car crash footage and noticed that cars don't have the rigid body you think they do. They bend and twist a lot. So we decided to put character rigs in the cars so they'd flex organically."

The performance of Maya's internal renderer, which handled all the vehicles, debris and rain surprised Burrell.

"It's not as robust as many; it doesn't have all the bells and whistles of mental ray or RenderMan. But it was really cool to see how much control we had with Maya's internal renderer."

"Continuity was a big deal throughout this film," Burrell emphasizes. "It was important to always focus on the story and continuity following basic film rules. With the greenscreen footage, we had no points of reference about where the actors were – there was no indication of north, south, east, or west. So we had to develop continuity. That the temp effects we did early on for Robert were so close to the finals in many ways helped get us in that direction."

Burrell maintained continuity at CaféFX by "keeping a running edit in our edit bay. I'd render out QuickTime scenes and drag them onto the desktops of the artists so they could keep an eye out for things like moon placement which always had to be in the western sky: off-camera glows could come from other angles."

In the end, CaféFX succeeded in creating a world for "The Big Fat Kill" that is only suggested by Miller's graphic novels. Quite an achievement when you consider that last summer the only thing they had in hand was footage of actors in front of greenscreens.

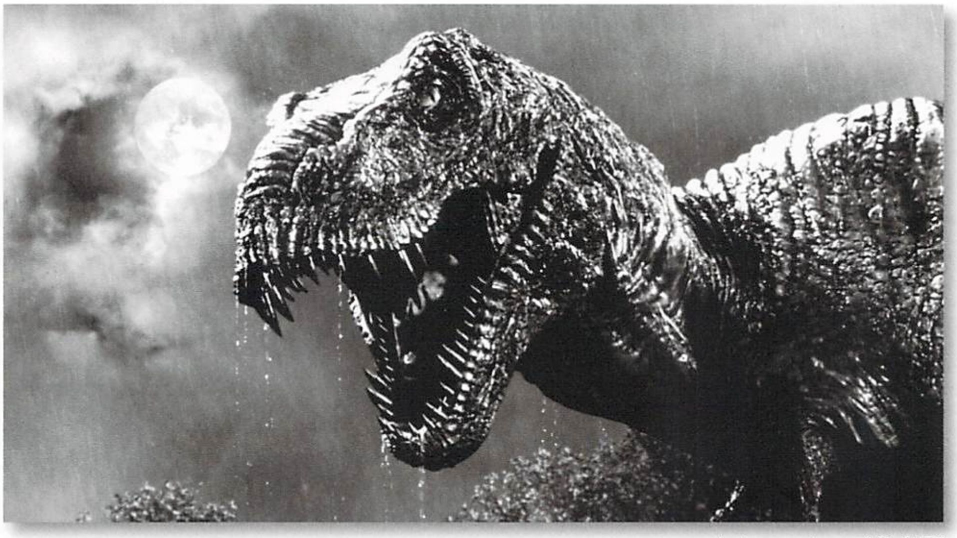


photo courtesy of CaféFX

Christine Bunish *has been writing about the film and video industries for most of her career. After holding staff positions at Broadcast Management/Engineering, World Broadcast News and Millimeter magazine, she became a freelance writer and editor. Her work appears in Post, Markee, Scene to Screen (UK), Profiles (UK), Tv y Video (Latin America), Advertising Age, Television Week and Media Business. She was also a contributor to the three-volume Advertising Age Encyclopedia of Advertising. She resides in Cedar Grove, New Jersey.*

SIN CITY "THE BIG FAT KILL" CAFÉFX VISUAL EFFECTS CREDITS

Visual Effects Producer
Edward Irastorza

Digital Effects Supervisors
Everett Burrell, Jeff Goldman

CG Supervisor
David Lombardi

Sequence Producer
Steve Dellerson

Art Director
Peter Lloyd

CG Supervisors
John Parenteau, David Ridlen

CG Lead Artists
Vlad Bina, Mike Fischer

3D Digital Artists
Steve Arguello, Lee Carlton, Domenic DiGiorgio, Gregg Domain, Alex Friderici, Phillip Giles, Victor Grant, Manuel H. Guizar, Trevor Harder, Grzegorz Jonkajtys, Szymon Masiak, Luke McDonald, Daniel Naulin, Will Nicholson, Mark Norrie, Brett Paton, Peter Profetto, Nic Spier, Rob Tesdahl, Gabriel Vargas

Compositing Leads
Mike Bozulich, Tom Williamson

Compositors
Jeff Allen, Michael Breymann, Daniel Bryant, Doug Cram, Christina Drahos, Brian Fisher, Scott Gordon, Michael Kennen, Sean Kennedy, Votch Levi, Ed Mendez, Shelly Morrow, Leah Nall, Greg Nelson, Kym Olsen, Akira Orikasa, Jennifer Scheer, Roman Ziad Seirafi, Anh Vu, Radost Yankova

Rotoscope Supervisor
Derek Krauss

Rotoscope Artists
Kevin Coyle, Loring Doyle, Joe Hoback, Randy Little, Josh McGuire, Toby Newell, Ruben Rodas, Eddie Soria, Melissa Widup

VFX Editors
Kevin LaNeave, Desi R. Ortiz

VFX Assistant Editor
Jimmy Lillard

Production Coordinators
Dawn Brooks, Maxine Jurgens

Production Support
Phillip Moses

Production Assistants
Jessica Becker, Wendy Hulbert, Tyler Williams

Render Wranglers
Brian Openshaw, Bernardo Rodriguez

IT Support
Jack Wells, Larry Thomas, Daniel Torres

Office Personnel
Kathi Galloway, Vange Ingan, Rhonda Thompson, Charmaine Tuason

Accounting
Sharron Sever

Executive Producer
Vicki Galloway Weimer

Production Executives
Jeff Barnes, David Ebner, O.D. Welch

"The revolutionary nature of ZBrush leads to Weta Digital creating a new approach to modeling hero creatures, digital doubles and props for film 3 in the trilogy. Throughout our work on "Lord of the Rings" we have been extracting displacement maps from high-resolution geometry. For films 1 and 2 we were obliged to use 3D scans of physical maquette to obtain this high resolution geometry, a time consuming and expensive process, because software didn't exist that could sculpt geometric detail to the level required."

"Now with ZBrush that software is available and in many cases we are replacing the maquette scan with a "digital maquette" that is sculpted on the computer. With support for interactively updating models of up to 4 million faces, we are now able to add details such as muscle and fat definition, skin and cloth wrinkles, and surfaces such as rock, cast iron, and weathered timber."

"Without the delay and cost of sculpting and scanning a physical maquette we have more time to refine the look of our models, and the software has the necessary flexibility to work in a production environment with changes to art direction and the level of detail required."

~ Matt Aitken, Digital Models Supervisor at WETA Digital

ZBRUSH

2D & 3D PAINTING, MODELING & TEXTURING

www.ZBrush.com ~ www.ZBrushCentral.com

The Ultimate Artistic Solution

Join us at Siggraph August 2-4, 2005

Pixologic Booth #1411

Pixologic™

Patents Pending ~ © 1997-2005 Pixologic, Inc. all rights reserved ~ Pixologic and the Pixologic logo, ZBrush and the ZBrush logo are registered trademarks of Pixologic, Inc.

Final Preparations Prior to Binding the Model

PART 3

Part 3 in the series of Preparing a Human Model for Animation. The two Previous parts can be found in HDRI 3D Magazine, Issues 3 & 4

This is a continuation of the rigging tutorial that has been running consecutively for the past issues of *HDRI 3D* magazine. This lesson will discuss the setting up of the eyes, eyelashes, eyebrows, teeth, and tongue, the creation of Blend Shapes for facial expressions, binding the character to the rig, refining the skin point weights of the mesh to control the manner in which specific joints deform the mesh, and finally, using Blend Shapes to control unsightly deformations at the joints.

Once you are finished with this lesson, you should be able to place your model in complicated poses. Sitting poses, for instance, are difficult to achieve without some undesirable deformations. Creating the right balance of weight maps and Blend Shapes that control unwanted distortions will help you achieve the poses you desire.

THE WORKFLOW

After completing the rig for your model, you should set it up for facial animation before binding it to the skeleton. Your workflow should be in the following order:

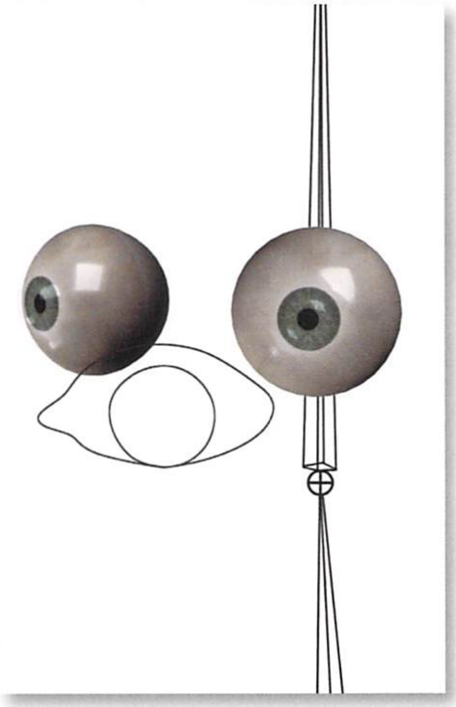
1. Modeling
2. UV mapping
3. Making the rig
4. Deleting history on the model
5. Eyeball movement with Aim Constraints
6. Creating the Blend Shapes

By Peter Ratner
Professor, Artist
Penn Laird, Virginia.

7. Creating Cluster Deformers for the eyebrows and eyelashes
8. Using Set Driven Keys to make the Blend Shapes drive the movement of the eyelashes and eyebrows
9. Making the open mouth Blend Shapes drive the rotation of the lower teeth and gums
10. Binding the skin to the skeleton
11. Editing the Skin Point Weights of the bound model
11. Controlling joint and muscle deformations with Blend Shapes.
12. Making hair
13. Animating
14. Rendering

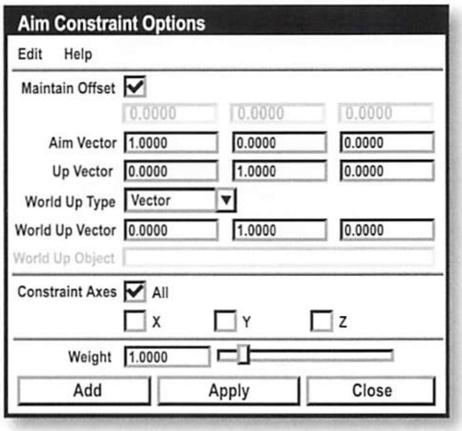
Notice the fourth step, which says to delete the model's history. When modeling, you accumulate a lot of history. In order to have the workflow progress smoothly, you need to delete this history, which takes up a lot of memory and can cause unpredictable things to happen. Once you have deleted the history, you should not do any more modeling or texturing, since this will begin to add new history to your model. You can push and pull existing vertices without creating any more history, but if you add or delete vertices, edges, and faces, you will accumulate history.

Delete the history on the low polygon mesh. The smooth proxy mesh should **not** have its history deleted or you will lose the connection between the two.



▲ *Fig. 1– Creating an eye icon with the EP Curve Tool and parenting each eyeball to the head joint.*

▼ *Fig. 2– Using an Aim Constraint to control the movement of the eyeballs.*



Maintain Offset ☑
0.0000 0.0000 0.0000
Aim Vector 1.0000 0.0000 0.0000
Up Vector 0.0000 1.0000 0.0000
World Up Type Vector
World Up Vector 0.0000 1.0000 0.0000
Constraint Axes ☑ All
☐ X ☐ Y ☐ Z
Weight 1.0000

CONTROLLING THE MOVEMENT OF THE EYES

In this section, you will learn how to set up an eye controller. It uses an icon with an Aim Constraint that makes the eyeballs follow the controller **(Figures 1 and 2)**.

STEP 1

Create an eye icon similar to the one in **Figure 1**. Use the EP Curve Tool (*Modeling > Create > EP Curve Tool*). Close the curve by going to *Edit Curves > Open/Close Curves.*) Position it in front of the model's face in-between the two eyes.

Select the eye icon and go to *Modify > Center Pivot.* While the eye icon curve is selected, go to *Edit > Delete By Type > History.* You should also select the eye icon and go to *Modify > Freeze Transformations* so that the Channel Editor only shows zero settings for translate and rotate. These properties make it easier to remember the eye icon's neutral position.

STEP 2

Place the eyeball, with its UV maps, inside the model's eye socket. Use the Smooth command (*Modeling >Polygons > Smooth > Options*) on the eyeball. Set the Subdivision Levels to either one or two, depending on how close the camera will be to the eyeball. You can always change this setting later in the Channel Editor if you do not delete the eyeball's history.

Copy and paste the eyeball and move it to the other eye socket. In the Channel Editor, change the Scale X to a minus setting so that it mirrors the orientation of the duplicate eyeball.

STEP 3

Make sure the local rotation axes of both eyeballs match those of the head joint. Select the left and right eyeballs first, and then Shift-select the head joint. Press "p" to parent the eyeballs to the head joint **(Figure 1)**. The last object selected before parenting becomes the parent. Test the rig by rotating the head joint. The eyeballs should move with it. Return the head joint back to its neutral position.

STEP 4

Select the eye icon EP curve and then Shift-select the outer transparent part of the left eyeball. Go to *Animation>Constrain>Aim>Options.* In the Aim Constrain option box, use the same settings as seen in **Figure 2**. Turning on Maintain Offset keeps the eyeball from spinning around. Select the eye icon EP curve and Shift-select the inside part of the right eyeball that has the UV texture map. Go to *Animation>Constrain>Aim.* Repeat this process for the right eyeball. Test the rotation of the eyeballs by moving the eye icon around. Be sure to undo the movement after you finish experimenting.

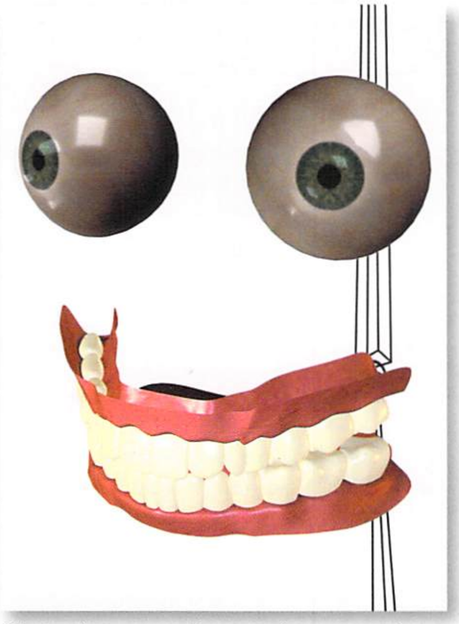


Fig. 3– The separate lower teeth, upper teeth, tongue, and gums are parented to the head joint.

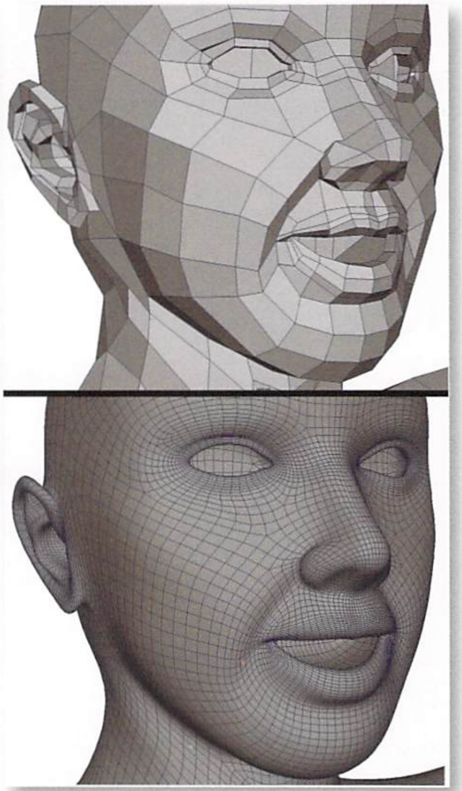


Fig. 4– The first Blend Shape (open smile) is modeled on the neutral character. Placing the smooth proxy version next to the low poly model makes it easier to see which vertices need to be moved.

SETTING UP THE TEETH AND TONGUE

Check the rotational axes of the tongue, and the lower and upper teeth. They should face the same direction as the head joint: Z-axis to the front, Y-axis up and down, and X-axis to the sides. Combine the lower teeth and lower gums as one object. The same goes for the upper teeth and gums. The tongue will have its own independent freedom of movement, so it will be a separate object.

Parent the lower teeth and gums to the head joint by selecting each of them and then Shift-clicking the head joint and pressing "p" on the keyboard. Select the upper teeth and gums and then Shift-select the head joint and press "p." Do the same with the tongue **(Figure 3)**.

MAKING BLEND SHAPES FOR FACIAL EXPRESSIONS

Maya allows you to deform a mesh with a series of target meshes that you manipulate in an editor with sliders. These are Blend Shapes (sometimes known as "morphs"). These Blend Shapes are useful for tasks such as changing facial expressions during an animation.

The most flexible part of the human face is the mouth. It is capable of forming many different expressions; therefore, the mouth will have the most Blend Shapes. Of course there are other methods to make facial expressions such as using Joints, Cluster Deformers, and Set Driven Keys. The advantage to using Blend Shapes rather than these other methods is that it can be difficult to achieve specific expressions without actually modeling them. Set Driven Keys also take up more memory than Blend Shapes. Furthermore, it is easier to model expressions by moving the vertices on the face rather than indirectly sculpting them with bones (joints) or clusters.

You should always make Blend Shapes for facial expressions before binding the skin to the skeleton. Adding a new Blend Shape node after binding forces you to reorder the deformers so that the skinCluster node is at the top.

Be sure that you fully texture your model and that its form requires no more fine-tuning before making any Blend Shapes. You can texture the model later, but complete your UV maps before starting the Blend Shapes. The UV maps actually make it easier to select specific parts of the model, such as the lips, and then hide what you are not working on at the time. Do not delete the model's history once you start making Blend Shapes because this will delete them.

STEP 1

If you have not done so already, delete the model's history prior to making Blend Shapes, because you will not be able to delete the history from now on.

STEP 2

Your model should be in the typical "T Pose" with its arms outstretched. It should remain in this exact pose and position the entire time. Select the low polygon version of the smooth proxy model. Export the

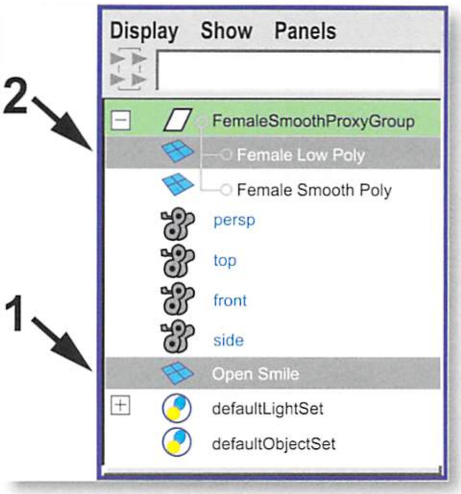

2
1
Display
Show
Panels
FemaleSmoothProxyGroup
Female Low Poly
Female Smooth Poly
persp
top
front
side
Open Smile
defaultLightSet
defaultObjectSet

Fig. 5– Selecting the first Blend Shape target in the Outliner and then holding down the Control key while selecting the low poly version of the model.

model as either an .obj or .mb file. Name it "Neutral Pose."

If your computer has enough RAM, open Maya again so that you have two separate Maya scenes open. If you do not have the computer memory for Maya to be open twice, then just open the neutral body in a new scene.

If you exported the file in .mb format, delete its history, and in the Outliner, delete everything except for the mesh. Also, get rid of any extra layers carried over from the previous file.

On the "Neutral Pose" file, move points to make the first Blend Shape **(Figure 4 -** *Previous page***)**. You can move vertices, but you should not add any new ones to your model. The Blend Shape target is the model on which you make the Blend Shape. Both the Blend Shape target and your original model should have the same point count.

You can make things easier for yourself by selecting the low poly neutral model and going to *Window > UV Texture Editor...*. In the UV Texture Editor, right-click and select "Face." Continuing in the UV Texture Editor, draw a selection around the mouth area faces. Go back to your Perspective view and select *Show>Isolate Select>View Selected* from its view menu. You can now work on shaping the mouth without having to deal with other parts of the body. Select *Show>Isolate Select>View Selected* again when you want to see the entire model.

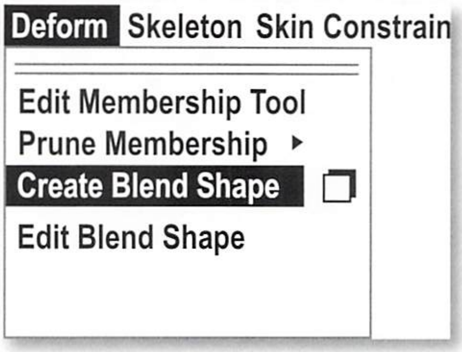
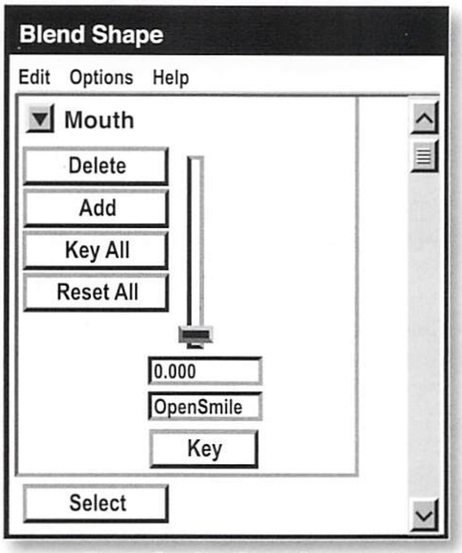

Deform
Skeleton
Skin
Constrain
Edit Membership Tool
Prune Membership
Create Blend Shape
Edit Blend Shape

Fig. 6– Executing the Create Blend Shape command.

In this exercise, we're modeling all the mouth Blend Shapes first. Since it is difficult to judge the final appearance of the smooth model by viewing only a low polygon version of it, make a new smooth proxy of the first Blend Shape target so that you can see what the actual high resolution Blend Shape will look like. When you finish sculpting the first Blend Shape, delete the smooth version and save the low polygon Blend Shape target with a different name, such as "Open Smile." The unaltered neutral pose will remain in its original state so that it can be used to make all the rest of the Blend Shapes.

STEP 3

In your scene file that contains your original character and its rig, go to the File menu and import the first Blend Shape target that you just completed. In the Outliner, select this Blend Shape Target and name it "OpenSmile." Continuing in the Outliner, hold the Control key down and select the original low poly proxy **(Figure 5)** object. Go to *Animation > Deform > Create Blend Shape > Options* **(Figure 6)**. In the Create Blend Shape Options box, next to Blend Shape Node, type "Mouth" and click the Create button.


Blend Shape
Edit
Options
Help
Mouth
Delete
Add
Key All
Reset All
Key
Select

0.000

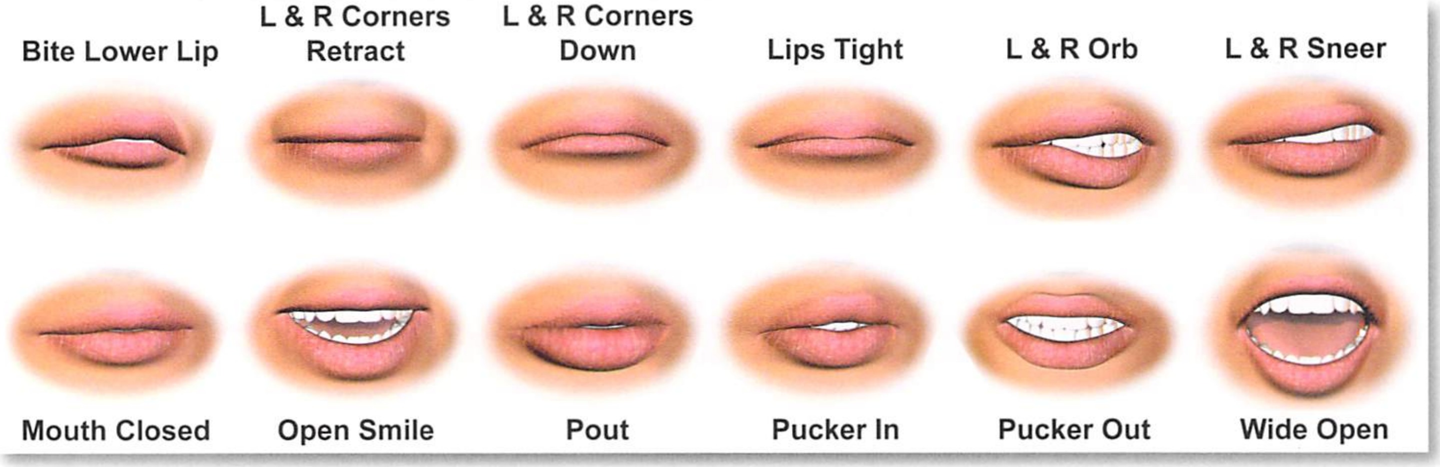
OpenSmile

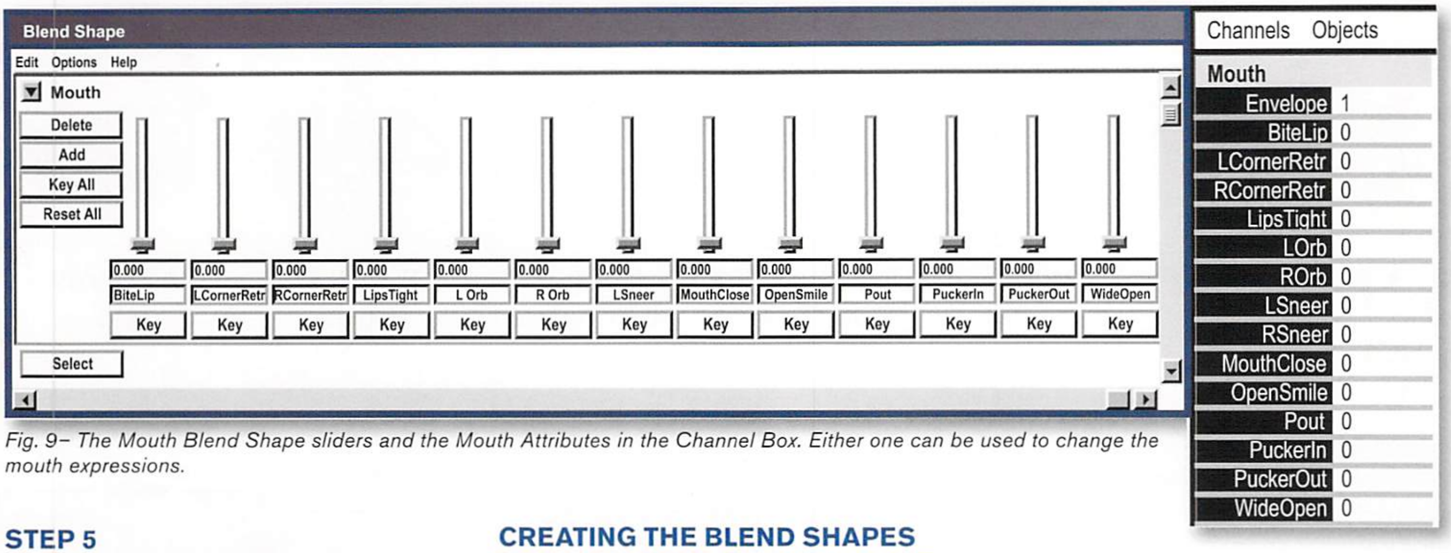
Fig. 7– The first Blend Shape under the "Mouth" heading (node).

STEP 4

You will not notice anything until you go to *Window > Animation Editors > Blend Shape*. In the Blend Shape box, you should now see a slider that you can move up and down to change the mouth expression **(Figure 7)**. If you didn't name the Blend Shape target, you can still change the name under the slider so that you can tell your Blend Shapes apart from each other. If you want to make any alterations to the Blend Shape, then do so on the imported Blend Shape target, not the original model. If you make changes to this target, then export it and overwrite the original "Open Smile" Blend Shape target. You can then delete the Blend Shape Target to keep the file size down. Check your original file size without the Blend Shape against the one with the "Open Smile." The file size should have increased only by about 5K.

Fig. 8– The basic mouth Blend Shapes. By moving the sliders in various combinations, these should handle most dialogue and facial expression tasks.


Bite Lower Lip
L & R Corners Retract
L & R Corners Down
Lips Tight
L & R Orb
L & R Sneer
Mouth Closed
Open Smile
Pout
Pucker In
Pucker Out
Wide Open



BiteLip	LCornerRetr	RCornerRetr	LipsTight	L Orb	R Orb	LSneer	MouthClose	OpenSmile	Pout	PuckerIn	PuckerOut	WideOpen
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Envelope	1
BiteLip	0
LCornerRetr	0
RCornerRetr	0
LipsTight	0
LOrb	0
ROrb	0
LSneer	0
RSneer	0
MouthClose	0
OpenSmile	0
Pout	0
PuckerIn	0
PuckerOut	0
WideOpen	0

Fig. 9– The Mouth Blend Shape sliders and the Mouth Attributes in the Channel Box. Either one can be used to change the mouth expressions.

STEP 5

Open a new scene with the neutral body again and create the next mouth Blend Shape target. After modeling it, import it into your scene that contains your original model with its "Open Smile" Blend Shape and rig. In the Outliner, rename the imported Blend Shape target, and while selected, Control-select the original low poly proxy object.

Go to *Animation > Deform > Edit Blend Shape > Add > Options*. In the Add Blend Shape Target Options box, click on Specify Node. Next to Existing Nodes, select "Mouth." Click the Apply and Close button. You should now see another slider next to the first one under the "Mouth" heading.

Continue making all the mouth Blend Shapes in this way. You can refer to **Figure 8** for all the mouth Blend Shape targets. Many Blend Shapes have individual targets for the right and left parts of the face. When the illustrations say "L & R," it means to model them separate from each other.

Be sure to look in a mirror when you make the Blend Shapes. You can also look up the Peter Levius site at: *www.3d.sk/*

The mouth Blend Shapes can now be activated either through the Blend Shape box with sliders or in the Channel Box by selecting the name Mouth under INPUTS **(Figure 9)**. Middle-click/drag after selecting one or more of the mouth Blend Shapes in the Channel Box.

These mouth Blend Shapes should be enough for most facial expressions and dialogue because you can mix them by moving the various sliders in different combinations. If you find that after binding you need a few more, then make additional ones and go to *Animation > Deform > Edit Blend Shape > Add > Options*. On the other hand, if you create a new node such as Ears, then you will have to reorder the deformers so that the skinCluster node is at the top.

CREATING THE BLEND SHAPES FOR THE BROW, EYELIDS, NOSE, AND CHEEKS

The following steps will make Blend Shapes for changing the brow, eyelids, nose, and cheeks. Later, weighted Clusters and Set Driven Keys will make the eyebrows and eyelash hairs follow the movement of the brow and eyelid Blend Shapes.

STEP 1

If possible, keep the scene file that has your model with all the mouth Blend Shapes and rig open. Open Maya again and import or open the neutral body model. Model the first eye Blend Shape for the left closed eyelid. Since this is one of the blink Blend Shapes, only the upper eyelid vertices rotated or move down. **Figure 10** shows the left blink as both the low polygon and smooth versions. Select the low polygon model, export it as either an .obj or .mb file, and name it "LeyeBlink."

STEP 2

In your original scene that has your model with the rig, import the "LEyeBlink" Blend Shape. In the Outliner, select this Blend Shape Target and name it "L_Blink." Continuing in the Outliner, hold the Control key down and select the original low poly proxy object. Go to *Animation > Deform > Create Blend Shape > Options*. In the Create Blend Shape Options box, next to Blend Shape Node, type "Eyes" and click the Create button.

Before you delete the "L_Blink" Blend Shape target, make sure that you are satisfied with the way the eyelid closes on the low and smooth proxy models. Once you delete the Blend Shape target from the scene, you will no longer be able to edit the Blend Shape.

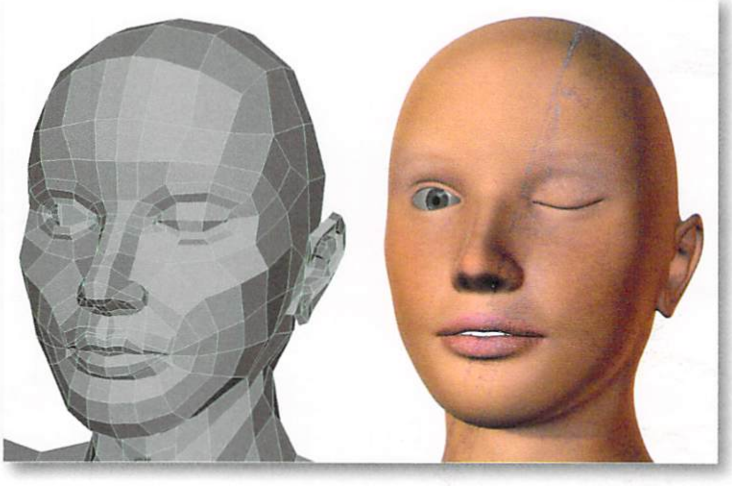
STEP 3

After deleting the "L_Blink" Blend Shape target, go back to the other scene file and open the neutral body model again. Model the right eyeblink Blend Shape target.

STEP 4

Go back to the original file and import the right eyeblink Blend Shape target. Rename it "R_Blink," Control-select the low proxy model, and go to *Animation > Deform > Edit Blend Shape > Add > Options*. In the Add Blend Shape Target Options box, click on Specify Node. Next to Existing Nodes, select "Eyes." Click the Apply and Close button.

Fig. 10 The first eye area Blend Shape target named "L_Blink"



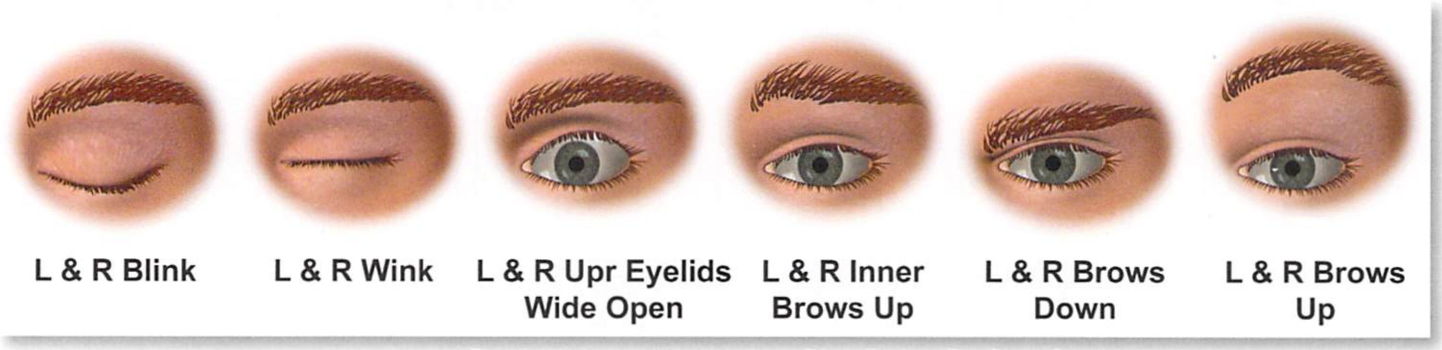
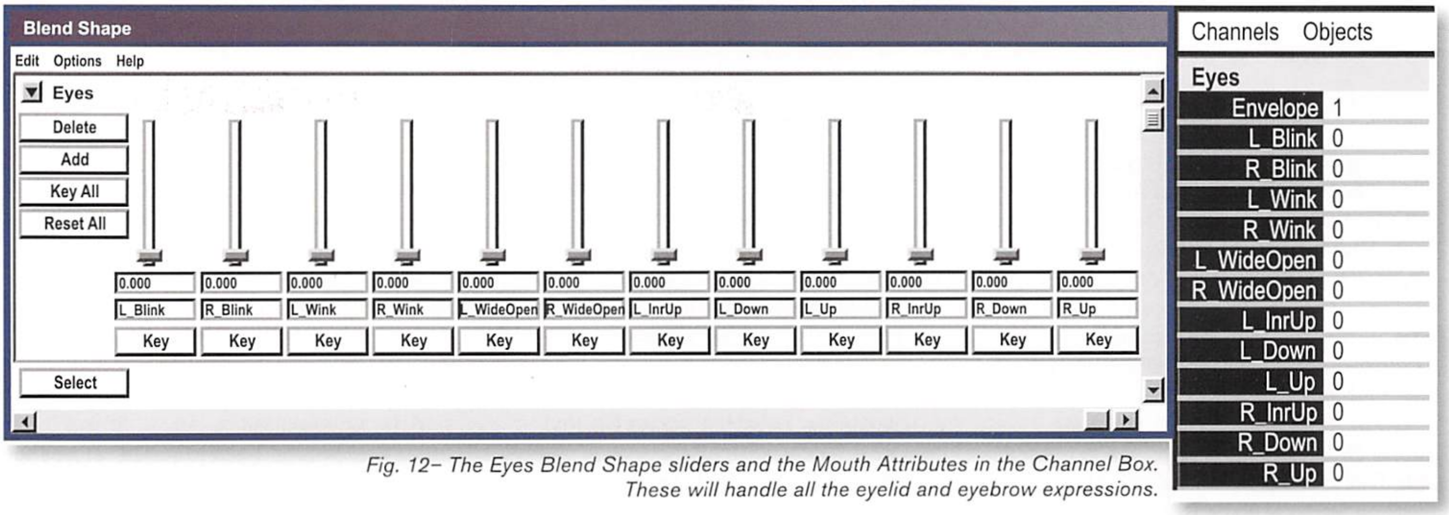

L & R Blink
L & R Wink
L & R Upr Eyelids Wide Open
L & R Inner Brows Up
L & R Brows Down
L & R Brows Up

Fig. 11– All the eyelid and eyebrow Blend Shapes. The Blend Shapes are modeled only on the polygon faces, not the eyelashes and eyebrow hairs, which are just seen here as reference.


Blend Shape
Edit
Options
Help
Delete
Add
Key All
Reset All
Key
Select
Channels
Objects

Eyes

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L_Blink	R_Blink	L_Wink	R_Wink	L_WideOpen	R_WideOpen	L_InrUp	L_Down	L_Up	R_InrUp	R_Down	R_Up

Eyes

Envelope	1
L_Blink	0
R_Blink	0
L_Wink	0
R_Wink	0
L_WideOpen	0
R_WideOpen	0
L_InrUp	0
L_Down	0
L_Up	0
R_InrUp	0
R_Down	0
R_Up	0

Fig. 12– The Eyes Blend Shape sliders and the Mouth Attributes in the Channel Box. These will handle all the eyelid and eyebrow expressions.

STEP 5

Continue making all the Blend Shapes for the eye area and brow. Refer to **Figures 11 and 12**, which illustrate all the Blend Shape targets. We're not dealing with the eyelash and eyebrow hairs right now, but seeing them here makes it easier to distinguish between the Blend Shapes.

The difference between the blink and the wink is that during the wink, the upper and lower eyelids meet in the middle, rather than just the upper eyelid going all the way down.

Instead of just modeling the eyebrows going up and down, the inner part of the eyebrow (near the nose) lifts up for puzzled and worried expressions. The next group of Blend Shapes is for the cheeks and nose. Since there are not that many, they can be under one node titled "CheeksNose."

STEP 6

Follow the previous steps to create a new "CheeksNose" Blend Shape node and then add all the nose and cheek Blend Shapes to it. You can refer to **Figure 13** for the Blend Shape targets.

Experiment by combining the various Blend Shapes. See if you can create the six recognizable universal emotions: sadness, anger, joy, fear, disgust, and surprise **(Figure 14)**.

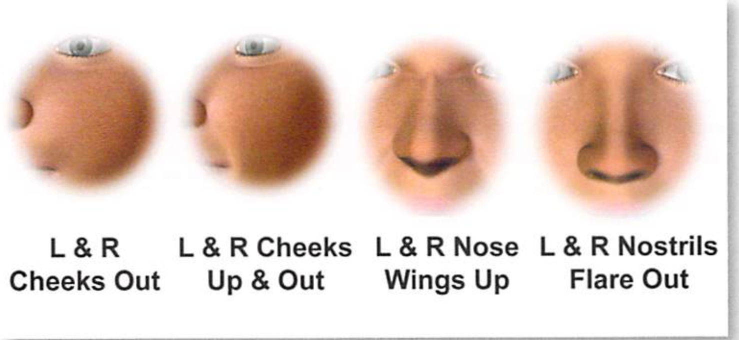
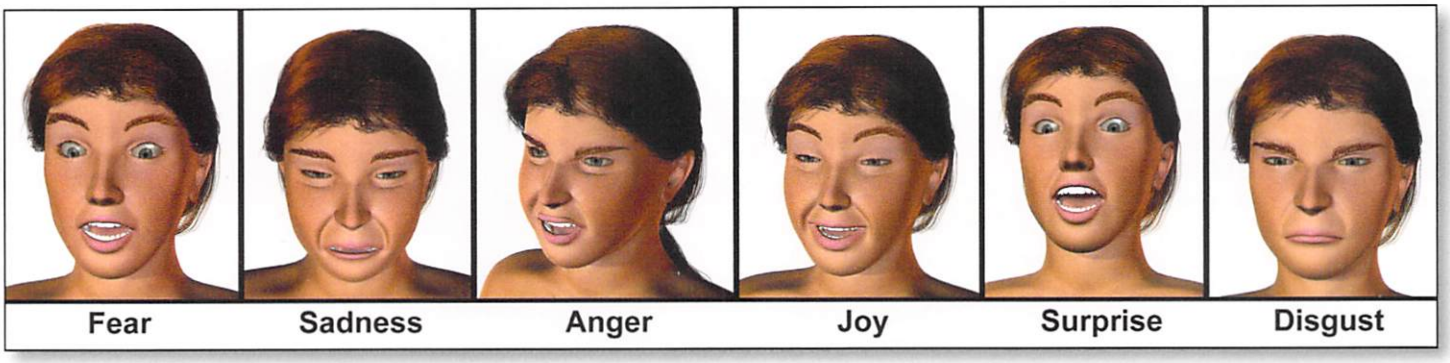

L & R Cheeks Out
L & R Cheeks Up & Out
L & R Nose Wings Up
L & R Nostrils Flare Out

Fig. 13– The cheek and nose Blend Shapes. Each side is modeled as a separate Blend Shape target.

Fig. 14– The six universally recognized emotions created by mixing the various Blend Shapes.


Fear
Sadness
Anger
Joy
Surprise
Disgust

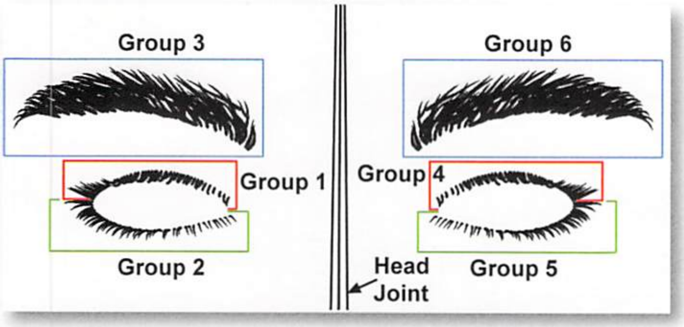


Fig. 15– Step 1. Grouping the eyelashes and eyebrows and parenting the six groups to the head joint. Each eyelash and eyebrow hair is left as a separate object.

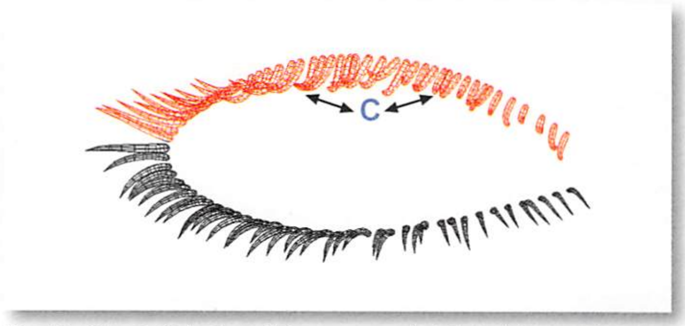


Fig. 16– Step 2. Selecting all the upper eyelash hair objects and making a cluster for them.

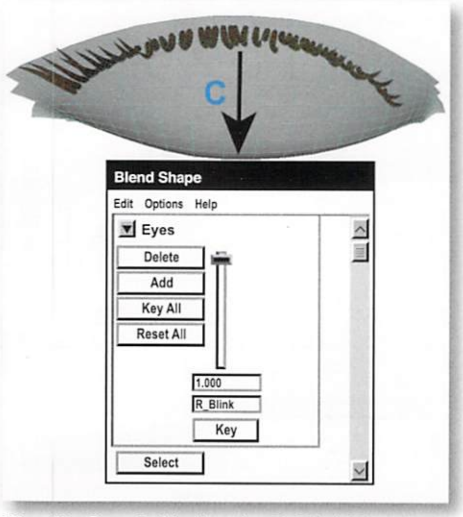
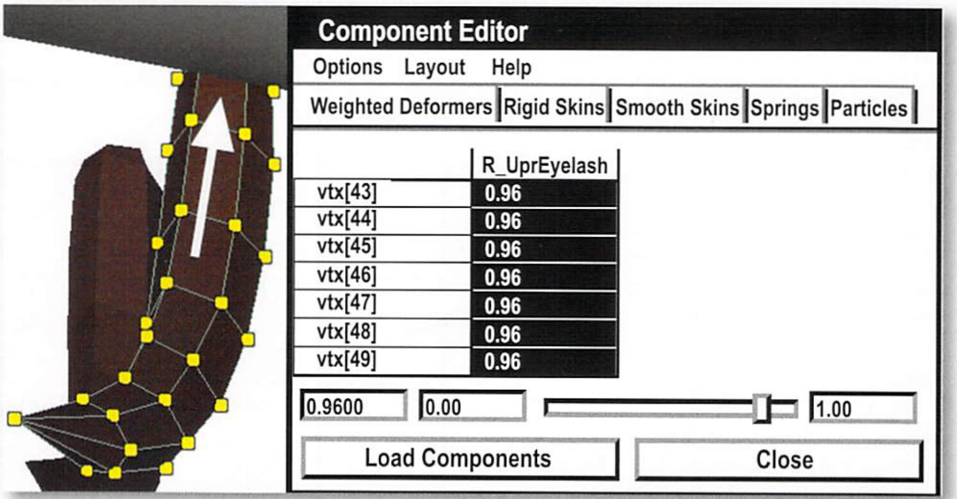


Fig. 17– Step 3. Moving the R_Blink Blend Shape slider all the way up to close the right upper eyelid. At this time, the upper eyelashes are still in their original up positions.



	R_UprEyelash
vtx[43]	0.96
vtx[44]	0.96
vtx[45]	0.96
vtx[46]	0.96
vtx[47]	0.96
vtx[48]	0.96
vtx[49]	0.96

0.9600 0.00 1.00

Fig. 20– Step 6. After Shift-selecting all the weight values of the vertices and opening the Component Editor, the slider is moved to the left until the eyelash lines up with the lowered upper eyelid.

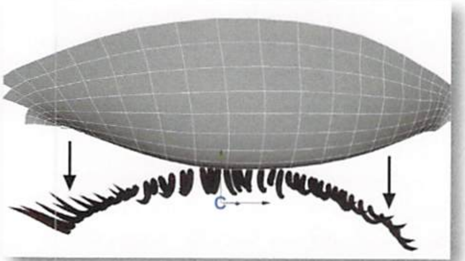
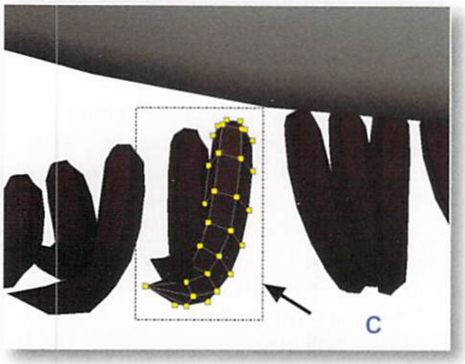


Fig. 18– Step 4. Moving the upper eyelash cluster down also displaces the upper eyelashes.

Fig. 19– Step 5. Selecting all the vertices of one eyelash hair.



If you plan to use a lip-sync tool such as Magpie Pro, then it is better to create another Blend Shape node for all the Visemes (speech mouth shapes) by moving sliders on the Mouth Blend Shape node. Each time you form a Viseme this way, export the model as an .mb or .obj file, and then import these into your scene and make the Blend Shapes to add to your new Viseme Blend Shape node.

CREATING AND EDITING EYEBROW AND EYELASH CLUSTER WEIGHTS

Now we'll make the eyelashes and eyebrows conform to the deformation. You need right and left upper and lower eyelashes, plus separate left and right eyebrows. Do **not** combine the individual eyelash and eyebrow hairs.

STEP 1

Separate the upper right eyelashes and lower right eyelashes into four separate groups **(Figure 15)**. Parent the four groups of eyelashes to the head joint.

STEP 2

Select all the upper right eyelashes and make a cluster for them by going to *Animation > Deform > Create Cluster* **(Figure 16)**.

STEP 3

Move the Blend Shape slider all the way up to close the right upper eyelid (R_Blink). Hide everything except for the smooth right upper eyelid faces, the right upper eyelash cluster, and the right upper eyelashes **(Figure 17)**. An easy way to do this is to make Quick Select Sets of these (*Create > Quick Select Set...*), Control-select them in the Outliner, right-click on the names, and pick "Select Set Members." Go to the Perspective view window menu and select *Show > Isolate Select > View Selected.*

STEP 4

Select the right upper eyelash cluster and check the Channel box to make sure that its Translate X, Y, and Z-axes are set to 0. Move the cluster down to move the upper eyelashes into the same position as the lowered eyelid **(Figure 18)**, and adjust their weights so that they all line up correctly.

STEP 5

Select one eyelash at a time, and press F8 to go into Vertex Selection Component mode. Draw a selection around all the vertices of the single eyelash **(Figure 19)**.

STEP 6

Go to *Window>General Editors>Component Editor...* Select the Weighted Deformers tab. Shift-select all the values for the vertices of that eyelash. On the bottom of the Component Editor window, locate the interactive slider and move it to the left until the eyelash is in the right position **(Figure 20–previous page)**.

STEP 7

Leave the Component Editor open and continue adjusting the weights of the other eyelashes until all of them line up correctly along the lowered eyelid **(Figure 21)**.

STEP 8

Move the cluster handle until all the upper eyelashes line up correctly along the smoothed lowered eyelid. You will most likely have to move the cluster handle along the Z-axis as well as the Y-axis. Write down the translate values as seen in the Channel Box of the cluster handle **(Figure 22)**. Move the cluster handle back to its original 0 translate position.

STEP 9

Open the Set Driven Key box and load the cluster in as driven with the translate values selected in the right column. Load the Eyes Blend Shape as the Driver and select the R_Blink Blend Shape in the right column **(Figure 23)**. Move the Blend Shape slider back to 0. Press the key button. Move the Blend Shape slider all the way up to close the right upper eyelid. Select the cluster handle for the right upper eyelashes, and in the Channel Box, type the translate settings that you wrote down earlier so that the cluster moves the eyelashes down and lines them up with the lowered eyelid. Press the key button. Now when you move the R_Blink Blend Shape slider up and down, the individual eyelashes move right along with the eyelid.

STEP 10

Select the group of left upper eyelashes. Create a cluster handle for them. Assign the same weight values for each individual eyelash just like you did with the right upper eyelashes. Follow Step 9 to create Set Driven Keys for the left upper eyelashes. Use the same method to move the lower eyelashes by first moving the Wink Blend Shape slider all the way up.

The eyebrows should have two cluster handles on each eyebrow. The reason for this is that the first eyebrow cluster can move the eyebrow up and down with Set Driven Keys. The second cluster has its own weight values for the same eyebrow hairs. These weight values make the inner part of the eyebrow hairs curve up for puzzled expressions. Create the up and down movement with Set Driven Keys for the left and right, up and down eyebrow Blend Shapes and the first cluster. Load in the second cluster and create new Set Driven Keys for the Blend Shape called "L Inner Brows Up" and "R Inner Brows Up."

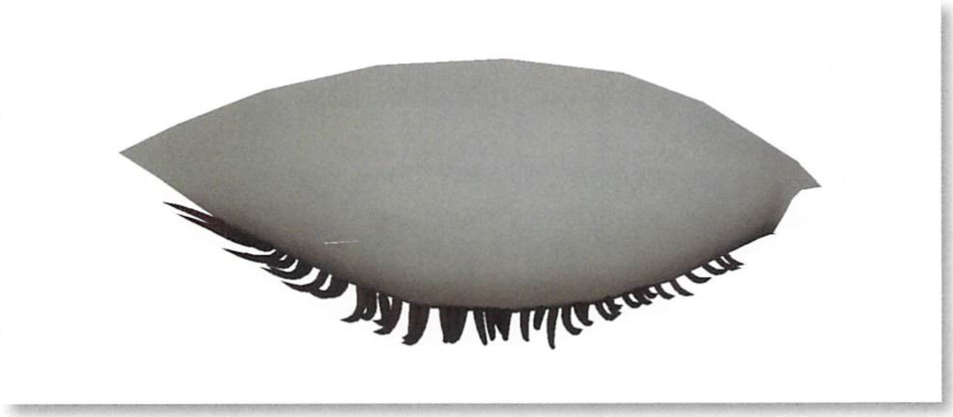
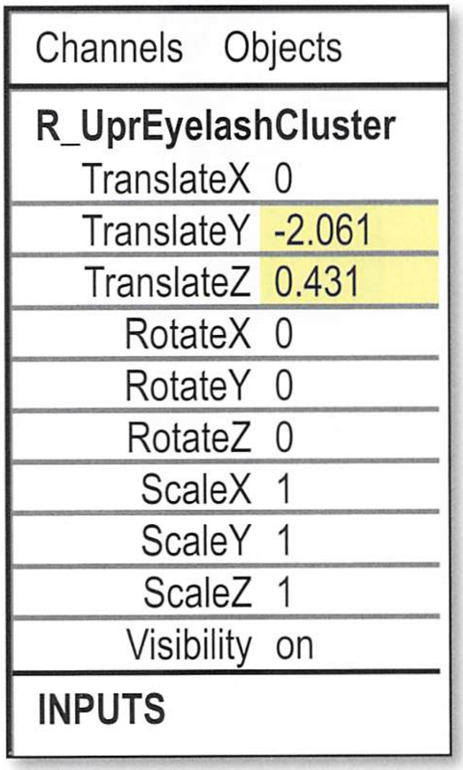


Fig. 21– Step 7. Once the individual eyelashes have the right weight values they will line up correctly with the bottom edge of the lowered eyelid.



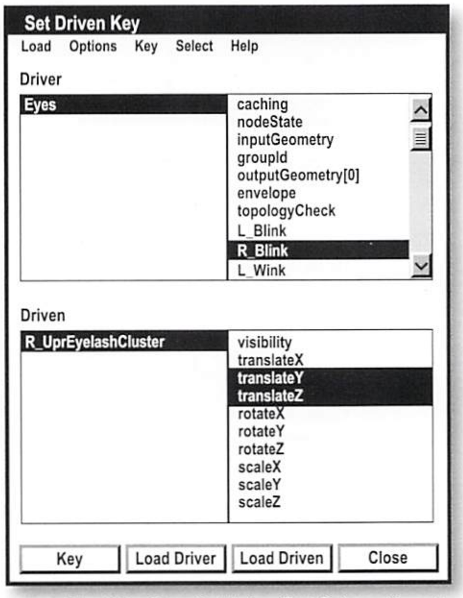
Channels Objects

R_UprEyelashCluster

TranslateX 0
TranslateY -2.061
TranslateZ 0.431
RotateX 0
RotateY 0
RotateZ 0
ScaleX 1
ScaleY 1
ScaleZ 1
Visibility on

INPUTS

Fig. 22– Step 8. After moving the right upper eyelash cluster on the Y and Z-axes, write down the values shown in the Channel Box.



Set Driven Key

Driver: Eyes — R_Blink

Driven: R_UprEyelashCluster — translateY, translateZ

Fig. 23– Step 9. Using Set Driven Keys to make the R_Blink Blend Shape drive the right upper eyelash cluster, which in turn moves the eyelashes along with the eyelid.

USING BLEND SHAPES TO DRIVE THE ROTATION OF THE TEETH AND TONGUE

The mouth parts (tongue, upper, and lower teeth) are separate objects parented to the head bone. When a Blend Shape such as the one named "Wide Open" rotates down to open the mouth, Set Driven Keys will make the lower teeth, gums, and tongue rotate down also. The following steps will show how to achieve this.

STEP 1

Select the low polygon model that now contains a "Wide Open" Blend Shape under the "Mouth" node. Go to *Animation > Animate > Set Driven Key > Set > Options*. On the right hand side, in the Channel Editor, under the Input heading, select "Mouth." This is the Mouth node of the Blend Shapes. In the Set Driven Key box, click the Load Driver button. Under the Driver heading, select "Mouth." In the right column for Driver, select "Wide Open."

Select the lower teeth and gums and press the Shift key down and also select the tongue. Click the Load Driven button. Under the Driven heading, select both the lower teeth and the tongue. In the right hand column for Driven, select RotateX **(Figure 24)**.

Press the Key button to key the teeth and tongue in their neutral positions when the

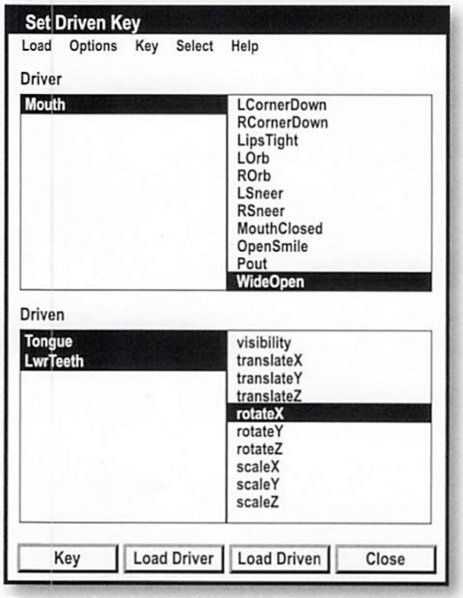

Set Driven Key
Load Options Key Select Help
Driver
Mouth
LCornerDown
RCornerDown
LipsTight
LOrb
ROrb
LSneer
RSneer
MouthClosed
OpenSmile
Pout
WideOpen
Driven
Tongue
LwrTeeth
visibility
translateX
translateY
translateZ
rotateX
rotateY
rotateZ
scaleX
scaleY
scaleZ
Key
Load Driver
Load Driven
Close

Fig. 24– Set Driven Keys are used to drive the teeth and tongue down when the Blend Shape "WideOpen" slider moves up.

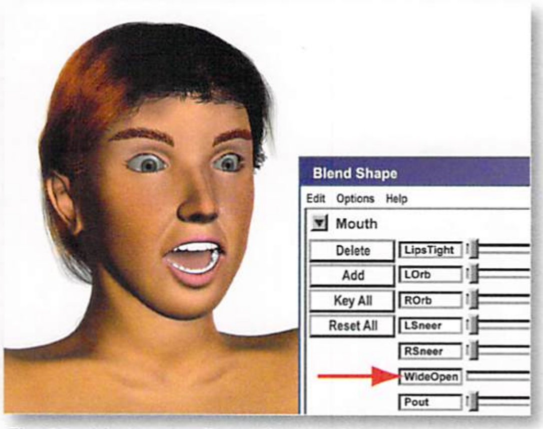

Blend Shape
Edit Options Help
Mouth
Delete
Add
Key All
Reset All
LipsTight
LOrb
ROrb
LSneer
RSneer
WideOpen
Pout

Fig. 25– Moving the Blend Shape slider to open the mouth also makes the lower teeth and tongue rotate down due to the Set Driven Keys.

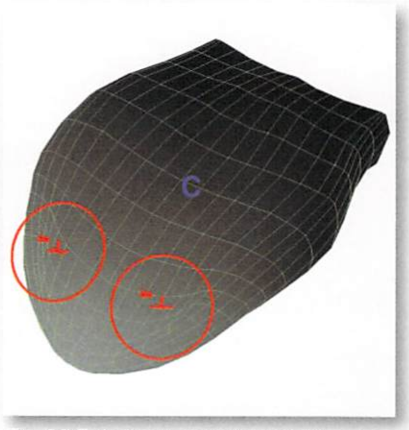


Fig. 26 Painting the cluster weights on the tongue.

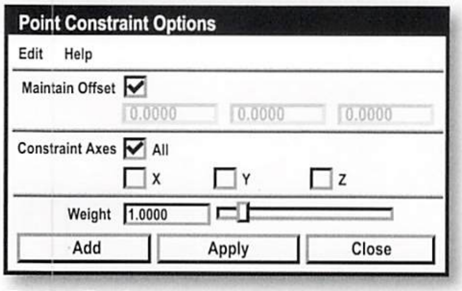
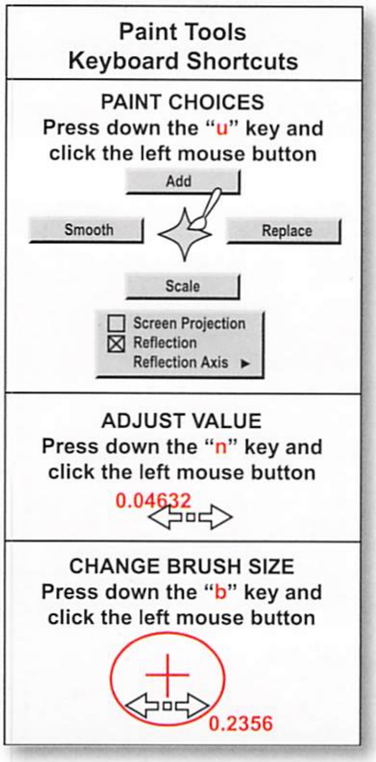

Point Constraint Options
Edit Help
Maintain Offset ☑
0.0000
0.0000
0.0000
Constraint Axes ☑ All
☐ X
☐ Y
☐ Z
Weight 1.0000
Add
Apply
Close

Fig. 27– The Paint tool keyboard shortcut options.

Fig. 28– After selecting the NURBS curve tongue icon, Shift-select the tongue cluster, and then open the Point Constrain Options box.


Paint Tools
Keyboard Shortcuts
PAINT CHOICES
Press down the "u" key and click the left mouse button
Add
Smooth
Replace
Scale
☐ Screen Projection
☒ Reflection
Reflection Axis ▸
ADJUST VALUE
Press down the "n" key and click the left mouse button
0.04632
CHANGE BRUSH SIZE
Press down the "b" key and click the left mouse button
0.2356

Copyright 2004-2005 Peter Ratner

mouth is closed. Open the Blend Shape box for the model and move the "JawDown" slider all the way up to open the mouth. Under Driven; select the lower teeth and tongue. Rotate them down on the X-axis. You might have to press the Insert key first to move the rotational pivot toward the back of the mouth more. Once you have the teeth rotated to where you want them when the mouth is wide open, press the Key button. Close the Set Driven Key box.

Use the slider in the Blend Shape box to open and close the mouth. The teeth and tongue should rotate along with the jaw when it moves down **(Figure 25)**.

If you ever get a Syntax Error when trying to apply Set Driven Keys, then change the name of the Blend Shape. In node and attribute names, all punctuation except for the underscore (_) and the pound sign (#) are illegal characters.

ADDING AN EXTRA CONTROL FOR THE TONGUE

We've established the tongue with Set Driven Keys, but there will be times when you'll want to have more direct control over it. The following steps show how to create a Cluster Deformer, adjust the weights, and then use a Point Constraint to move the tongue.

STEP 1

Make sure you select the layer containing all your Cluster Deformers. Click on the tongue and go to Animation > Deform > Create Cluster. Rename the cluster "TongueCluster."

STEP 2

Select the tongue again and go to Animation>Deform>Paint Cluster Weights Tool. Paint the back of the tongue with zero weights so that it does not move. Gradually lighten the gradation of the weight as it gets closer to the tip **(Figure 26)**.

Use these keyboard shortcuts to quickly work with the Paint Cluster Weights Tool **(Figure 27)**.

Click and hold the "b" key down to adjust the size of your brush.

Press the "u" key down and left-click to select the paint choices.

Hold the "n" key down and left-click to select the value of the Replace, Scale, or Add options.

STEP 3

Test the Paint Cluster weights by moving the cluster up and down and forwards. Adjust the weights if you need to do so.

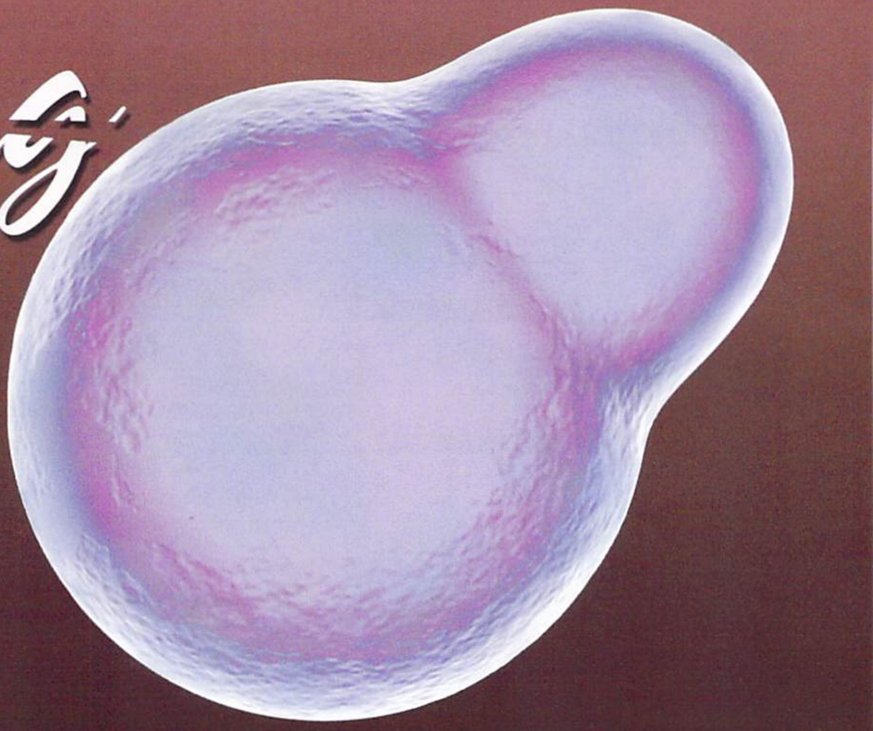
STEP 4

Create the NURBS curve icon used to move the tongue. Delete its History and Modify>Freeze Transformations. While selected, Shift-select the tongue Cluster. Go to Animation>Constrain> Point>Options. Use the same settings as seen in **Figure 28**. You should now be able to move the tongue with the NURBS curve icon.

This concludes all the preparation before finally binding the model to the rig. The next issue will show how to use both rigid and smooth bind for binding the model to the skeleton. The tutorial will also explain weight maps and several methods for adjusting them, as well as using Blend Shapes to control unsightly deformations at the joints.

Peter Ratner *is a professor of 3D Computer Animation at James Madison University. He is the founder and head of the first Computer Animation program in Virginia. His paintings, animations, and computer graphics have been displayed in numerous national and international juried exhibitions. He is the author of 3-D Human Modeling and Animation, 1st and 2nd Editions (John Wiley and Sons) and Mastering 3D Animation, 1st and 2nd Editions (Allworth Press). He lives in Penn Laird, Virginia.*

Splitting Cells



By Dan Ablan
3D Animator / Author
Founder– 3DGarage.com
Shaumberg, Illinois

I had a project that required an animation to show a splitting cell. There were a number of elements within this job, and the large pharmaceutical company was specific about what could be shown, and more importantly, what could not. In order to complete the task, LightWave's HyperVoxels came to the rescue. However, at the time of doing this project, I had to jump through quite a few hoops to accomplish the desired look. Why is this, you ask? Good question: not too long ago, when performing certain tasks with HyperVoxels, a project that required splitting cells was good only if you were using a basic, surface-based HyperVoxel. I need to use specific image mapped HyperVoxels because the cells in question had a distinct coloring, and consequently, the images did not blend well. The end result was not what the client wanted and it became a real problem. I worked around it using gradients, which you'll learn how to do in this project.

This tutorial will show you the steps to create your own splitting cell in LightWave using HyperVoxels. This is a variation of the project I created for my client. You'll create everything in Layout. **Figure 1** shows an example of two HyperVoxels with an image map applied; create the HyperVoxels with two null objects, and tell them to 'blend' within the HyperVoxel panel.

You can see that the image maps stay put on both nulls, which is great. In the past, the image would stretch across both images. With the latest version of HyperVoxels, this no longer happens. However, you can see that there is still an overlapping of the applied image. While the animation of the cells splitting works, the coloring from the image map does not. **Figure 2** shows the same two nulls without image maps, using all LightWave's texturing capabilities. Here, you can see the textures blend much better.

1) To create the image in **Figure 2**, start LightWave Layout and add a null object.
2) From the Windows dropdown menu at the top left of Layout, select Volumetric and Fog Options. This opens the Effects panel.
3) From the Add Volumetric selection within the Volumetric tab, choose HyperVoxels.
4) Double-click the HyperVoxel listing to open its controls. **Figure 3** shows the selection.

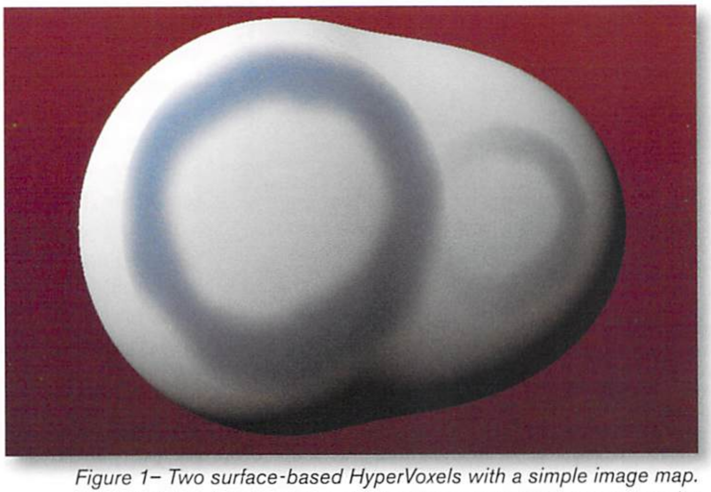
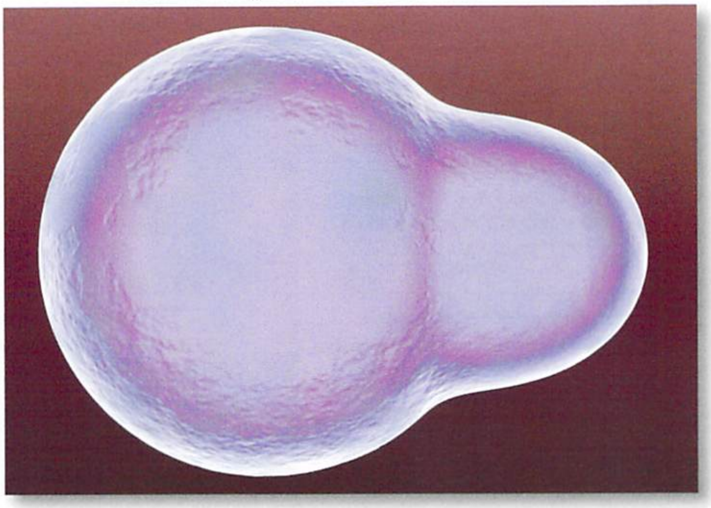
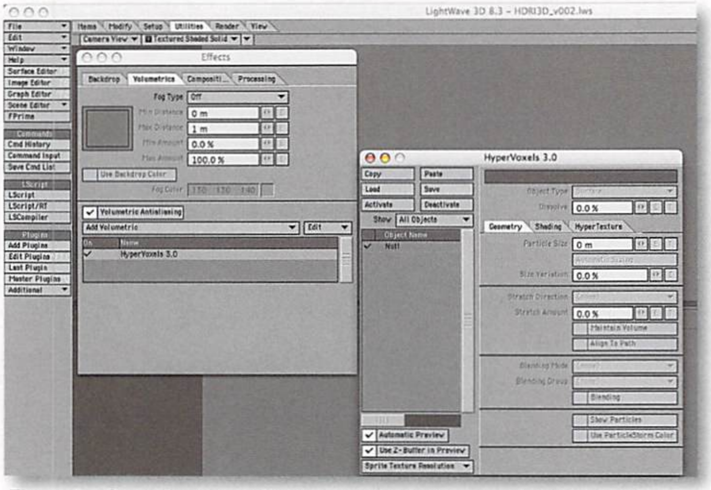


Figure 1– Two surface-based HyperVoxels with a simple image map. While the latest version of LW allows a single image map to be applied to both voxels, the images don't cross properly.

Figure 2– The same two surface-based HyperVoxels textured with gradients. The transition is clean and deliberate.





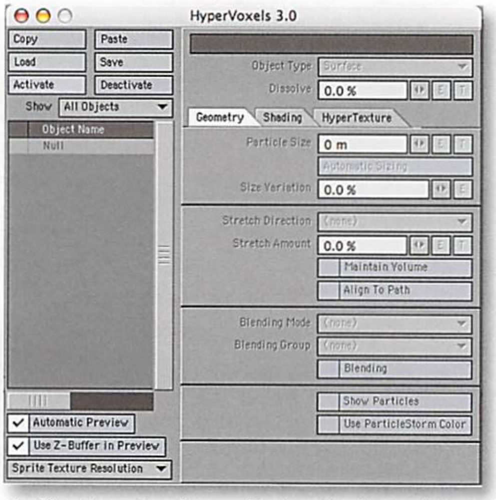
LightWave 3D 8.3 - HDRI3D_v002.lws

Effects

Fog Type: Off
Min Distance: 0 m
Max Distance: 1 m
Min Amount: 0.0 %
Max Amount: 100.0 %
☑ Volumetric Antialiasing
Add Volumetric
☑ HyperVoxels 3.0

File
Edit
Window
Help
Items
Modify
Setup
Utilities
Render
View
Backdrop
Volumetrics
Compositing
Processing
Edit
HyperVoxels 3.0
Copy
Paste
Load
Save
Activate
Deactivate
Geometry
Shading
HyperTexture

Figure 3– Open the HyperVoxels panel by first adding the Volumetric plug-in and then double-clicking the listing.

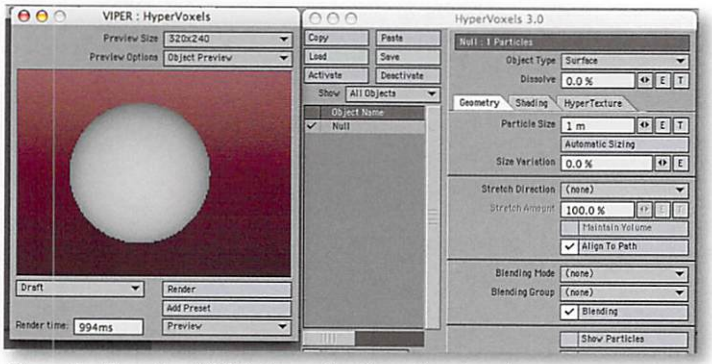


HyperVoxels 3.0

Show: All Objects
Object Name: Null
Object Type: Surface
Dissolve: 0.0 %
Particle Size: 0 m
Size Variation: 0.0 %
Stretch Direction: (none)
Stretch Amount: 0.0 %
Blending Mode: (none)
Blending Group: (none)
☑ Automatic Preview
☑ Use Z-Buffer in Preview
Sprite Texture Resolution

Copy
Paste
Load
Save
Activate
Deactivate
Geometry
Shading
HyperTexture
Automatic Sizing
Maintain Volume
Align To Path
Blending
Show Particles
Use ParticleStorm Color

Figure 4– The HyperVoxels panel before the first null is activated.



VIPER : HyperVoxels

Preview Size: 320x240
Preview Options: Object Preview
Draft
Render time: 994ms

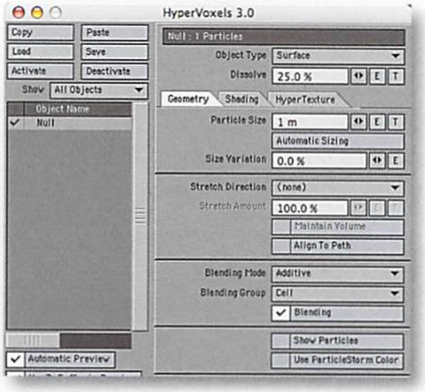
HyperVoxels 3.0

Null : 1 Particles
Show: All Objects
Object Name: ✓ Null
Object Type: Surface
Dissolve: 0.0 %
Particle Size: 1 m
Size Variation: 0.0 %
Stretch Direction: (none)
Stretch Amount: 100.0 %
☑ Align To Path
Blending Mode: (none)
Blending Group: (none)
☑ Blending

Render
Add Preset
Preview
Copy
Paste
Load
Save
Activate
Deactivate
Geometry
Shading
HyperTexture
Automatic Sizing
Maintain Volume
Show Particles

Figure 5– By using VIPER, the active HyperVoxel renders out.

Figure 6– Basic geometry settings for the first HyperVoxel out.



HyperVoxels 3.0

Null : 1 Particles
Show: All Objects
Object Name: ✓ Null
Object Type: Surface
Dissolve: 25.0 %
Particle Size: 1 m
Size Variation: 0.0 %
Stretch Direction: (none)
Stretch Amount: 100.0 %
Blending Mode: Additive
Blending Group: Cell
☑ Blending
☑ Automatic Preview

Copy
Paste
Load
Save
Activate
Deactivate
Geometry
Shading
HyperTexture
Automatic Sizing
Maintain Volume
Align To Path
Show Particles
Use ParticleStorm Color

5) The first thing you'll want to do when creating a splitting cell with HyperVoxels is open up VIPER so you can see what's going on! From the Render tab in Layout, click the VIPER button, or press F7. Then, from the HyperVoxels panel, double-click the null listing to activate it. Or, select it and press the Activate button, as in **Figure 4**.

> ***NOTE:***
> ***When working with VIPER, make sure the HyperVoxels panel is open in order to see a preview of your scene.***

6) As soon as you activate the null object within HyperVoxels, you'll see VIPER draw the image. If it doesn't, just click the Render button on the VIPER window. **Figure 5** shows your initial HyperVoxel cell.

7) Like any texture assignment, start modifying the settings for this HyperVoxel null from the top down. With HyperVoxels, you have three object type choices: Surface, Volume, and Sprite. The "object" is the null, or simply, a point. You can apply HyperVoxels to any point in Layout. A Volume HyperVoxel creates thick puffs, good for clouds, cotton, etc. A Sprite is a slice of the Volume HyperVoxel and renders much faster. Sprites are great for wispy smoke, dust, etc. For this project, we need to use a Surface-based HyperVoxel.

8) Set the Dissolve to 25% to give the object a slight bit of transparency. Next, in the Geometry tab of the HyperVoxels panel, Particle Size should be about 1m. Size Variation should be 0% because we're only working with one null object. If we had many nulls, or a clump of particles, setting a Size Variation would do just that – vary their sizes.

9) Because the cell should be round, leave Stretch Direction set to none. Change Blending Mode to Additive. Then, under Blending Group, select this option and choose New Group. Enter a name such as 'cell' and click OK. What you're doing here is creating a blend group so you can mix two HyperVoxels together. This essentially is the key to the splitting cell.

10) Finally, make sure Blending is checked. **Figure 6** shows the settings.

11) The next step is to create the surface of the cell. You can do this directly within the HyperVoxel panel. Click over to the Shading tab. You'll start with a basic color, a pale blue, about **064, 071, 102 RGB**. Next, click the T button to create a texture for the color surface. Initially, you're going to create a Gradient. Make the Layer Type a Gradient. Set Blending Mode to Normal if it's not already set, and leave Layer Opacity at 100%.

12) Next, set the Input Parameter to Incidence Angle. You're going to create four 'keys' in addition to the default key in the gradient bar. First, click about a third of the way up from the bottom to create the first key. Set the color to **54, 50, 95 RGB**. Change Alpha to 55% to dissolve this portion away slightly. The Parameter setting is simply the position of the key in the bar.

13) Add another key just about the one you first created, and set this to a deep purple color. Alpha is 100%. Then, create another deep blue key, also 100% Alpha. Create one more about a third of the way up from the top, and make its color light blue, about **145, 156, 204 RGB**. Alpha for this key is 100%. The last key should be all the way at the top (Parameter 0.0) and set to off white. **Figure 7** shows what the gradient should look like.

14) Keep an eye on the VIPER window to see how your gradient settings are applied. Click and drag the keys as you see fit to adjust the ring around the edge of the HyperVoxel cell. The Gradient you've created is the ring around the edge.

15) Add a new layer on top of the Gradient, but make it a Procedural Texture Layer. Set the Procedural Type to Turbulence, and make it a deep bluish green color, about **40, 58, 68 RGB**. Also, set Layer Opacity to 80% to dissolve the texture a bit. Down at the bottom of the settings, in the Scale tab, set the size to 200mm for the X, Y, and Z. This will simply add some noise on top of the cell so it's not so perfect in color.

16) If you like, add another Procedural Texture layer on top of the one you just created. Make its size about 80mm for the X, Y, and Z, and set the Layer Opacity to 50% or so. Change Frequencies to 1, and give it a deeper blue color. This will add even more variation to the cell color. **Figure 8** shows the settings.

17) Click Use Texture at the bottom of the panel. Save your scene. Now, from the Window dropdown in Layout, choose Presets, or press F8. This will open the Preset window where you can save these HyperVoxel settings. With VIPER open, double-click directly in the VIPER window. You'll see the same image appear as a thumbnail in the HyperVoxels Preset window. Now, anytime you want to use these texture settings for a HyperVoxel, simply select your HyperVoxel, and then double-click the Preset to apply.

18) Back out in the Shading tab of the HyperVoxel panel, change Luminosity to 200% and Diffuse to 150%.

19) One more setting to go and you're there. Click to the HyperTexture tab. This will allow you to create interest and apply cool displacements to your object. Change the settings to the following:

Texture	**Hetero Terrain**
Increment	**0.487**
Lacunarity	**2.1**
Octaves	**6.11**
Offset	**0.359**
Nose Type	**Perlin Noise**
Texture Amplitude	**2.0%**
Scale	**1m, XYZ**

20) Take a look at your VIPER window and you'll see how the HyperTexture settings create little bumps around the cell. Of course, feel free experimenting with these values. **Figure 9** shows the panels.

21) Go ahead and double-click the VIPER window to save the HyperVoxel setting to your Presets.

22) Back in Layout, save the scene. Now, there are a few more steps and you'll have a splitting cell! Select the original null object in the scene and press Ctrl C to clone it. Choose 1 for the number of clones and click OK.

23) Select the cloned cell and move it over slightly on the X-axis, away from the original null. If AutoKey is not on, create a keyframe for this null's new position at frame 0.

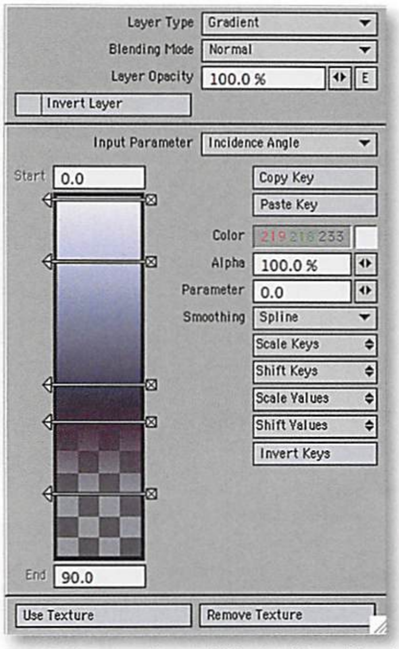
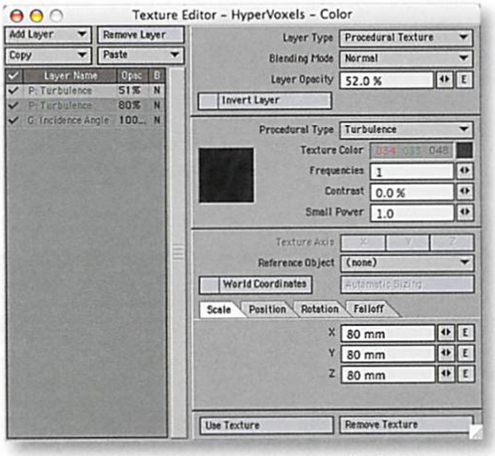

Layer Type: Gradient
Blending Mode: Normal
Layer Opacity: 100.0 %
Input Parameter: Incidence Angle
Start: 0.0
End: 90.0
Color: 219 218 233
Alpha: 100.0 %
Parameter: 0.0
Smoothing: Spline

Figure 7– The gradient settings within the HyperVoxel panel help create a soft ring around the outside edge of the cell.

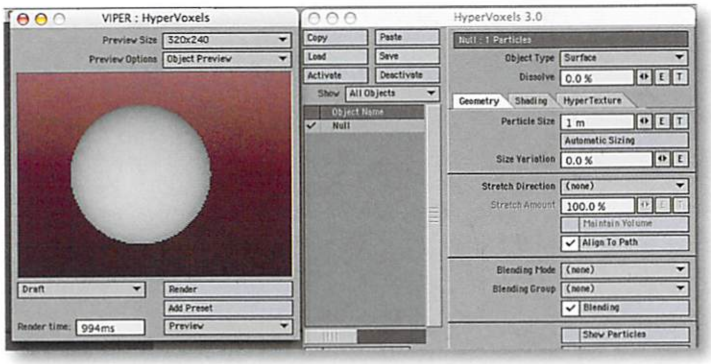

Texture Editor - HyperVoxels - Color

✓	Layer Name	Opac	B
✓	P: Turbulence	51%	N
✓	P: Turbulence	80%	N
✓	G: Incidence Angle	100...	N

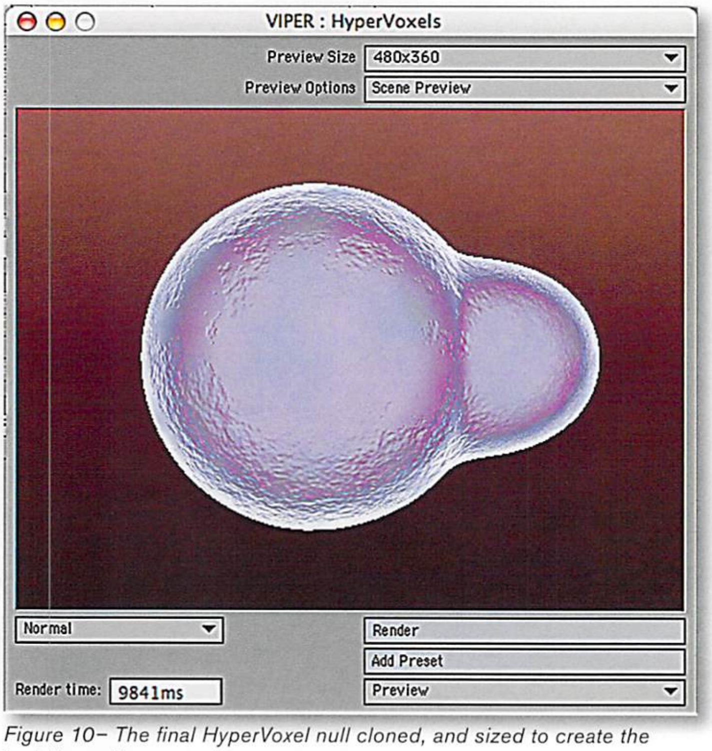
Layer Type: Procedural Texture
Blending Mode: Normal
Layer Opacity: 52.0 %
Procedural Type: Turbulence
Texture Color: 034 035 048
Frequencies: 1
Contrast: 0.0 %
Small Power: 1.0
Reference Object: (none)
X: 80 mm
Y: 80 mm
Z: 80 mm

Figure 8– Additional texture layers applied on top of the gradient in the HyperVoxels panel help add interest to the cell.

Figure 9– A HyperTexture is added to the cell's surface. As you can see from the VIPER preview, the cell now has small imperfections and bumps throughout.


VIPER : HyperVoxels
Preview Size: 320x240
Preview Options: Object Preview
Draft
Render time: 994ms

HyperVoxels 3.0
Show: All Objects
Object Name: ✓ Null
Null : 1 Particles
Object Type: Surface
Dissolve: 0.0 %
Particle Size: 1 m
Size Variation: 0.0 %
Stretch Direction: (none)
Stretch Amount: 100.0 %
☐ Maintain Volume
☑ Align To Path
Blending Mode: (none)
Blending Group: (none)
☑ Blending
☐ Show Particles


VIPER : HyperVoxels

Preview Size 480x360

Preview Options Scene Preview

Normal

Render time: 9841ms

Figure 10– The final HyperVoxel null cloned, and sized to create the budding cells.

24) With VIPER and the HyperVoxels panel open, you can see how the cells look. At the top of the VIPER panel, change Preview Options to Scene Preview. This will allow you to see both null objects at the same time.

25) Select Size from the Modify tab, and size down the second null about 40%. **Figure 10** shows the VIPER window with the null moved to the right, and sized. Voila! Splitting cell.

26) To create the animation, move the second null back to the position of the first. Create a keyframe at 0. Keep its smaller size. Now, at say frame 120, move the second null to the right and create a keyframe. At frame 200, size the second null back up to 1.0, its original size. You can make a preview in VIPER to see the animation and what you'll see is the cell's wall pushing out, and then splitting.

Even on a simple level, HyperVoxels are very powerful. We created something complex, a splitting cell, with nothing more than two null objects. Now, take this information and think big, and what you can create is mind blowing. Remember that you can always add more textures, more shading, and vary the look for endless possibilities.

***Dan Ablan** is president of AGA Digital Studios, Inc. in Chicago, IL., and founder of 3D Garage.com. He has written eight books for New Riders Publishing on 3D and computer graphics. Dan is the former editor of Keyframe and HDRI 3D Magazine.*

electric rain® swift3D®

Ultimate 3D Tools for Macromedia Flash®

electric rain swift3D 4.5

Swift 3D v4.5 is the industry leading tool for quickly and easily creating high quality 3D vector and raster animations for Flash®, video and print design projects.

Standalone
3D Flash Tool
$229.00

electric rain swift3D MAX3

Swift 3D MAX for 3D designers using 3ds max™, Swift 3D MAX allows you to export 3D animations to Flash, directly from your high-end 3D application.

Vector Plug-in for
3ds max
$295.00

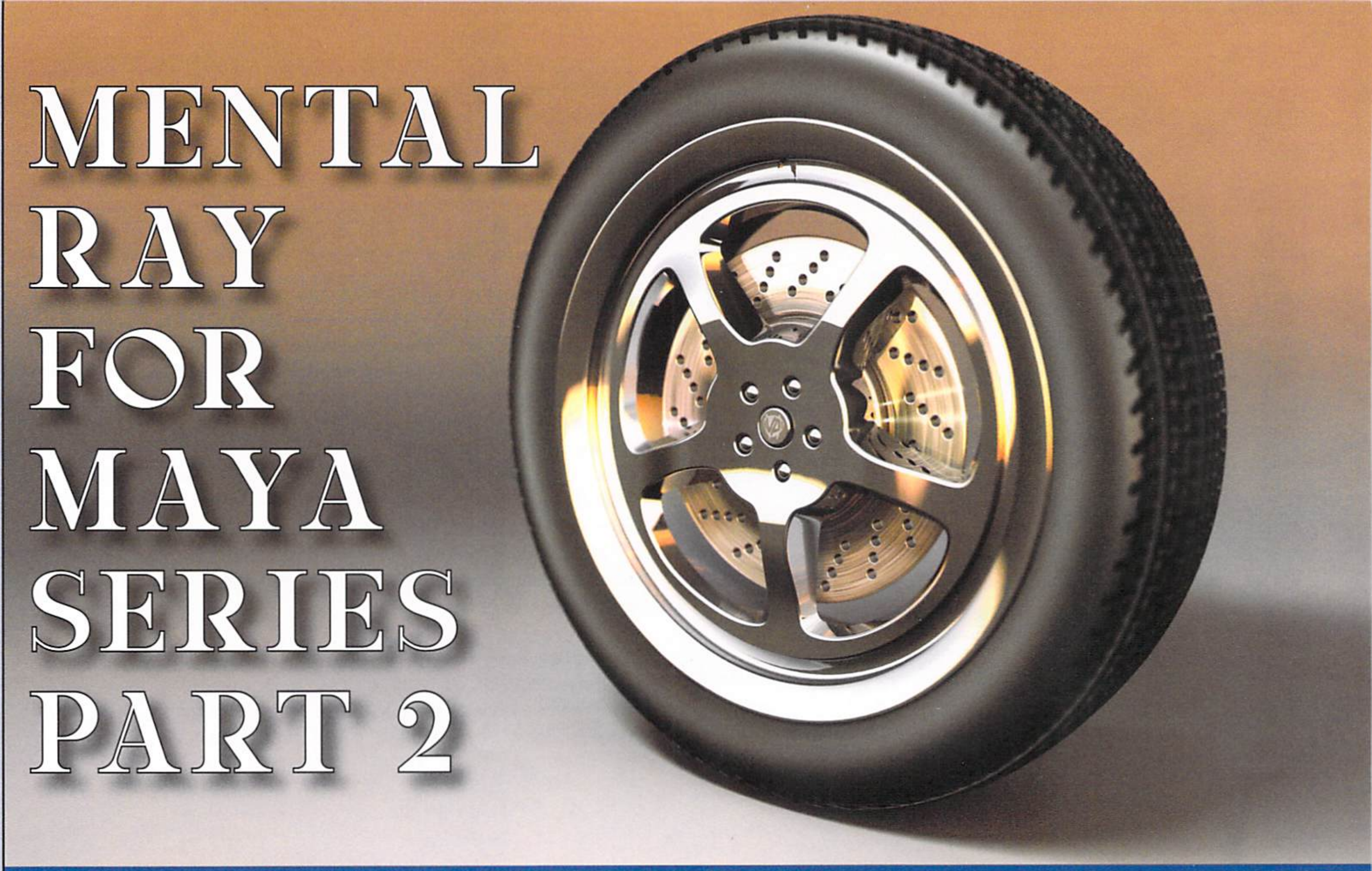
electric rain swift3D LW3

Swift 3D LW for 3D designers using LightWave® 3D, Swift 3D LW allows you to export 3D animations to Flash, directly from your high-end 3D application.

Vector Plug-in for
LightWave 3D
$295.00

Trial Download and more information available at www.erain.com

MENTAL RAY FOR MAYA SERIES PART 2



By Boaz Livny
3D Artist, Animator, Teacher
New York City, New York

RENDER PASSES WITH MENTAL RAY

This tutorial will examine custom render passes with Maya and mental ray; this includes an in-depth review of how color is evaluated during render time, and how we mimic similar evaluations with the rendered passes while compositing. The article follows up on the tutorial "Ambient Occlusion with mental ray" from HDRI Issue #4, as well as further explores the mathematical logic behind color values. In this article, I am focusing on render passes with Maya using mental ray, as well as the bit depth we use for different passes. The next article in this series for issue #6 will examine the compositing process for these passes using a node-based compositor.

RENDER PASSES – THEORY BEHIND THE COMPOSITE

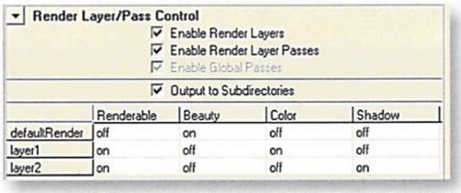
The ideal concept underlying rendering in passes is to divide the scene into as many elements as possible and to reconstruct them in a compositing package, enabling us to tweak each element.

The render passes make it possible to adjust specific attributes during the compositing process; for example, the Key Light passes can be tuned to match a general color scheme that is set during the compositing stage. In this case, having the Key Light passes separately provides control over the lighting in real time, during the composite, rather than re-rendering the 3D elements over and over.

When planning how to break up your scene, you must take into account your compositing resources / capabilities and create the most useful collections of passes for compositing; this might include separate passes for the Diffusive Color, Specular, Reflective, Shadow, Lighting, Occlusion, Environment Sampling, Normals, Z-depth, Y-gradients, and any other custom pass you may need.

MAYA'S RENDER PASSES VS. RENDER LAYERS

The Maya Render Layer/Pass Control in the Render Global Settings Window **(figure 1)**, under the mental ray tab, provides two completely different ways to render: one enables us to divide the scene into layers and render one user-defined layer at a time, while the other provides a way to control how the scene is rendered in Light passes. Ideally, you would combine the two for rendering your passes by layer as well as Light passes, although this may not currently be practical.



Render Layer/Pass Control

☑ Enable Render Layers
☑ Enable Render Layer Passes
☑ Enable Global Passes
☑ Output to Subdirectories

	Renderable	Beauty	Color	Shadow
defaultRender	off	on	off	off
layer1	on	off	on	off
layer2	on	off	off	on

Figure 1 – Render Layer/Pass Control

Render passes provide a way to render your scene in separate Color passes, such as Specular or Diffusive color, to later composite them to take advantage of the powerful resources available for adjusting each pass (such as color correction).

If you're not familiar with rendering in layers, from the layer window you can choose whether you want to work with the Display Layers or the Render Layers **(figure 2)**. The Render Layers applies at render time if the Enable Render Layers is enabled in the Render Globals, though the problem with this is that you have no interactive display for the layers in the view port, which leaves room for confusion.

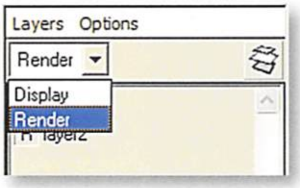


Figure 2 – Render / Display Layers

Ideally, separating the scene into individual rendered objects allows you to

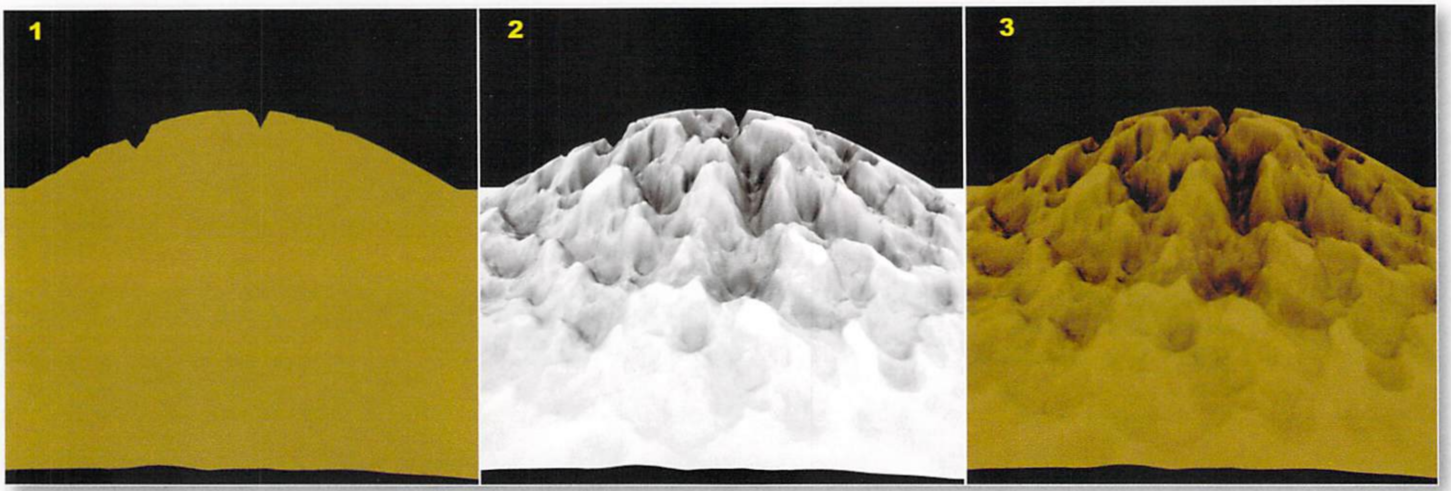


*Figure 3 – Color pass (1) * Occlusion pass (2) = Composite (3)*

combine and control them on a per object basis within a compositing package. The Render Layer / Pass Control provides more control over how the scene is rendered; unfortunately, it is not a means to an end. Rather then using this, I prefer to control my layers / passes by using separate scene files for each of the elements (or passes) and then run a batch render script to render all the separate scenes. You can always use a master scene using file referencing to ease this task and save out subsequent passes. This article and the upcoming article in Issue #6 demonstrate this workflow.

Personally, for what might be the leading 3D package in the industry, the render passes option in Maya is limited and almost useless; this should radically be changed to resemble rendering passes with SOFTIMAGE|XSI and mental ray as well as the Renderman render passes capabilities.

COLOR EVALUATION

When rendering a scene (without passes), the renderer determines the final color for each pixel by mathematically evaluating material and light properties, then outputting the result: the rendered image. Rendering is the process of calculating color based on mathematical evaluations of how the Diffuse, Specular, reflective, and any other color (such as light) contribute to each other. This process begins by determining the color value for every shaded point based on the surface shader attributes, and then examining other contributing factors, such as light or lens effects.

We examined how multiplying a grayscale Occlusion pass by the Color pass provides a means for controlling the surface color with respect to occluding objects (objects that partially block light) and Self Shadowing in Part One, Issue #04. Mathematically multiplying the grayscale Occlusion pass in **figure 3** (2) with the color pass in **figure 3** (1) results in a modified Color pass that has mathematically scaled the Color pass (1) values according to the Occlusion pass (2) values and provides for the result in **figure 3** (3). In short, if the Occlusion pass for a particular point is black (value equal to zero), multiplying an image by the color value for that Occlusion point results in a completely darkened point. On the other hand, multiplying an area by white (a value of one) results in maintaining the Color Value for that point, so there is no change from the Occlusion pass.

With Color passes, we add (mathematically) one color onto the other. For example, adding the Specular color onto the Diffuse color of the shader (Diffuse + Specular), and then multiplying the result by the light's color ((Diffuse + Specular) * light), creates the final rendered image. Recreating this sort of color evaluation with a composting tree will only be possible as long as we successfully separate each of these colors into their individual passes.

We understand, based on theory, that the render controls how colors are combined mathematically; thus, the right color plates and knowing the math allow us to recreate any render. Let's examine how we can recreate a basic render of a sphere with an attached Phong shader (blue Color Value and red Specular Color Value) and the default light **(figure 4)**.

When rendered, the Phong shader evaluation precedes the light evaluation, calculating color values for the surface first. The highlight color is determined by adding the Diffuse color with the Specular color to give a magenta hue for the Specular highlight, seen in **figure 4**. Specular attributes of the Phong shader define Specular color and behavior (falloff rate and color). Light direction and value help determine the highlight position and intensity.

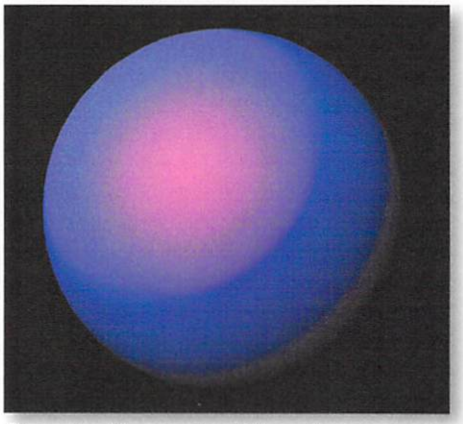


Figure 4 – Maya render of the sphere

Light, as in real life, defines form. Without light falloff, there would be no form and everything would appear flat, as with ambient light. During the render, the light (color and intensity) is multiplied against the sphere's color values (from the Phong shader); therefore, where the surface receives less light (smaller values), the sphere seems dark blue or even black, thus. light darkens colors that are multiplied by smaller values from the light's falloff.

Using color plates (single color images) that represent the Diffuse and Specular colors, as well as a grayscale gradient image, representing the light falloff and intensity **(figure 5)**, we recreate the render. Note that for this exercise, there are no rendered passes, only color images created within a 2D package. This emphasizes the evaluation of color and the role light plays.

Using After Effects, I created a composition with three-color plates from **figure 5**. I placed the blue plate at the lowest level, representing the Diffuse color. Next, I placed the red plate, which has an elliptical hold-out mask with some edge feathering, above the blue plate, representing the Specular falloff. I set the "add mode" for the red layer; this adds the red color onto the blue color, creating the magenta highlight color **(figure 6 - stage 1)**. This process has successfully recreated the evaluation used by the Phong shader for determining the Specular highlight color.

As you can see in **figure 6 - stage 1**, the image is flat, with no form. Defining form requires recreating the light falloff for a spherical surface. The grayscale gradient image resembles the light falloff for a spherical object with a circular gradient radiating out from the center. White represents the light color and intensity and black represents areas with no light. As the color gradually changes from white to black, the light influence decreases. I placed the gradient color plate above the previous two images in the composite and multiplied using the "multiply mode" **(figure 6 - stage 2)**.

Multiplying the gradient layer by both red and blue layers gradually darkens their color values as white moves to black, simulating the light falloff evaluation used by Maya during the render. A quick reminder for the underlying concept is, as white has a value of 1, when multiplied by the color value, the color value is preserved, and as white transitions to black the color is gradually multiplied by a decreasing value, eventually resulting with a zero color value, i.e. black.

Figure 7 shows the comparison between the rendered Maya sphere and the artificial sphere, created within After Effects. The only difference is the light directionality, which in Maya was not from the center. This RGB process has been converted to CMYK for print, so some noticeable color errors may occur because

RENDERING PASSES WITH MENTAL RAY

This section examines how to create custom passes that provide for maximum color separation, currently not offered by the Maya Render Layer/Pass Control **(figure 1)**. Using the master scene as a source, I created separate scene files for each pass I wanted to render, all of which have custom shaders and different render settings that will let me render exactly what I need for each pass. In a case like this, I recommend using a batch render script for rendering overnight, or using a render manager to queue the renders.

Figure 8 shows the image I will create, using the rendered passes composited with a node-based compositor. For clarity I divided the passes into three main sections, Color passes, Light passes and Custom passes. The following are technical explanations of how to achieve these passes. Note that I show more passes than I require for this image, to give a wider overview of creating proper passes.

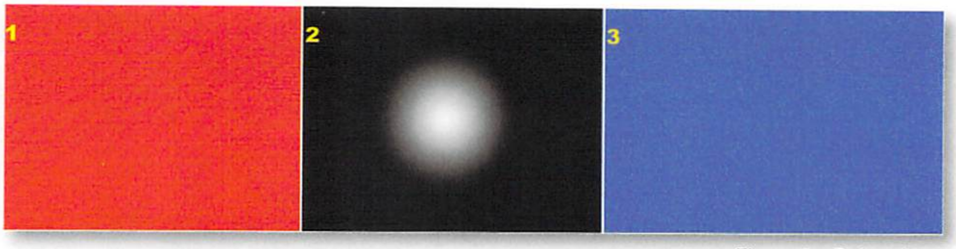
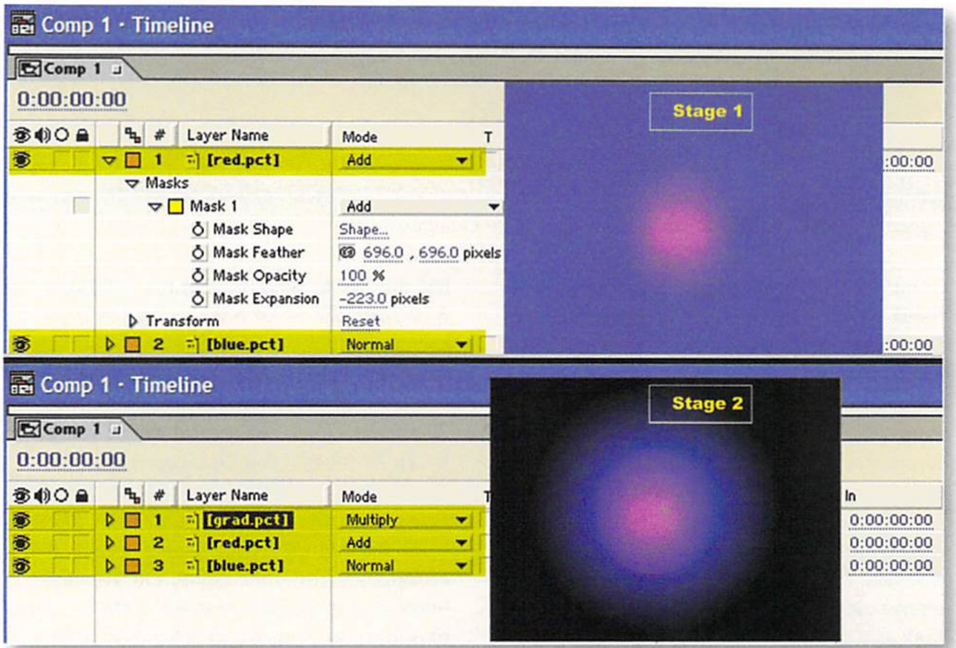


Figure 5– Color plates



Comp 1 · Timeline

Comp 1

0:00:00:00

#	Layer Name	Mode
1	[red.pct]	Add
	Masks	
	Mask 1	Add
	Mask Shape	Shape...
	Mask Feather	696.0 , 696.0 pixels
	Mask Opacity	100 %
	Mask Expansion	-223.0 pixels
	Transform	Reset
2	[blue.pct]	Normal

Stage 1

Comp 1 · Timeline

Comp 1

0:00:00:00

#	Layer Name	Mode	In
1	[grad.pct]	Multiply	0:00:00:00
2	[red.pct]	Add	0:00:00:00
3	[blue.pct]	Normal	0:00:00:00

Stage 2

Figure 6– Stage 1 & 2 for color composite.

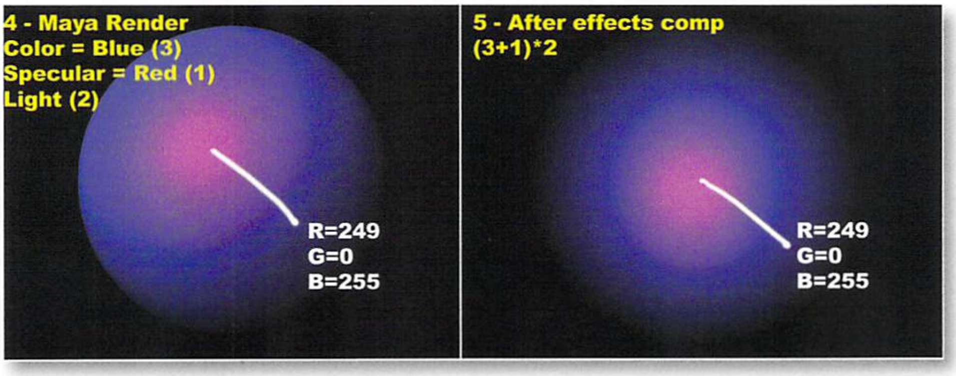
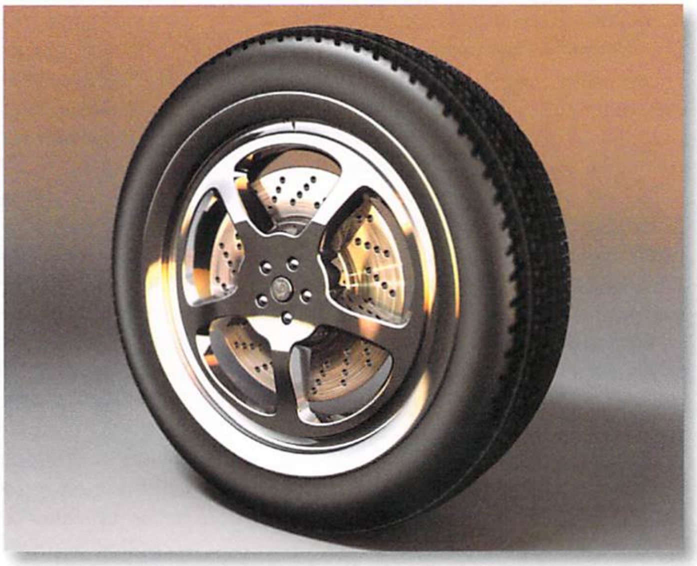


Figure 7 (ABOVE)– Color Evaluations comparison

Figure 8 (at RIGHT)– Image



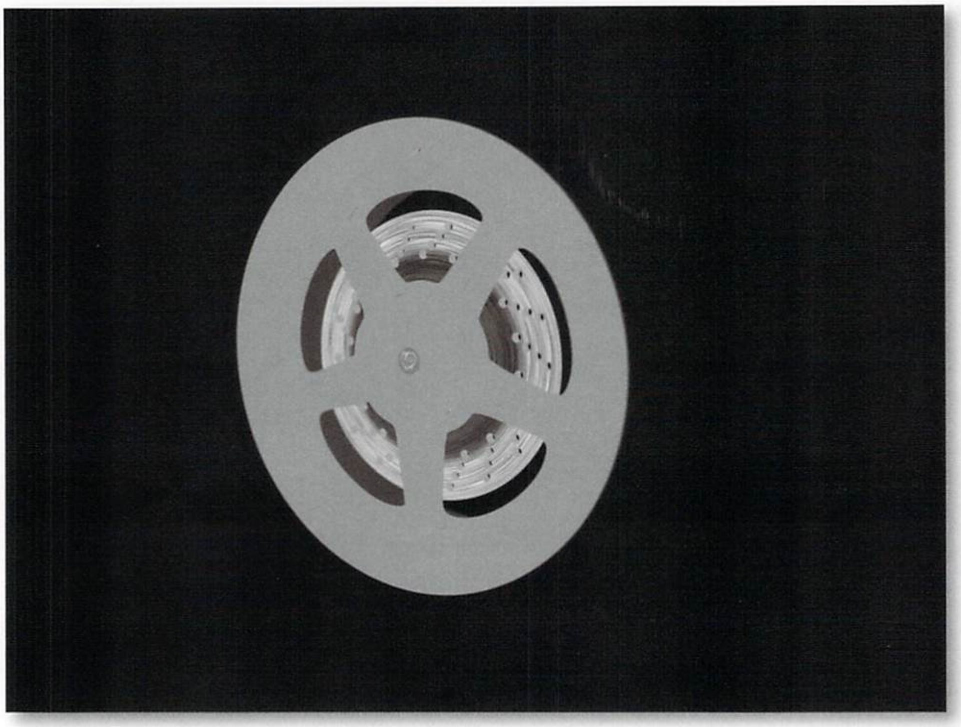


Figure 9 – Diffuse color pass

COLOR PASSES:

Diffusive Pass– Color pass

Framebuffer type: RGBA 8 or 16 Bit. **(see sidebar)**

Method: Maya passes using a surface shader or ambient light.

Select Enable Global Passes from the mental ray Render Layer/Pass Control, and then choose the Diffuse pass. The primary problem when using this pass is that it will also calculate the light's contribution. For rendering a Diffuse Color pass we want to solely extract the color values, without any light influence. To overcome this problem, in addition to enabling the Diffuse pass, you can do one of the following:

1 Create new shaders or convert each shader to the Surface Shader. Light does not influence the surface shader; it outputs the exact texture color. See Issue #4's first article in this mental ray series for more information on using the surface shader.

2 Delete or disable all lights (including disabling the default light option in the Render Globals), then create an ambient light with a zero Ambient Shade value. Reducing this value to zero gives an overall white ambient light to the scene with no shading. This solution works well for a large scene that contains a lot of materials, which you may not want to convert to the surface shader.

Set the Ambient Light Intensity and Color values to one and white, respectively; these settings ensure multiplication of the texture color by a Light value of one, which renders the correct Texture Color value. With an Ambient Light value of anything other than one, the texture color, influenced by the light, may shift in brightness as well as color.

Disable raytracing and shadows if the Diffuse pass option is not enabled. This optimizes the render time, as well as disallows the calculation of reflection color.

Specular Pass – highlight intensity

Framebuffer type: RGBA 8 or 16 Bit.

Method: Enable Specular only lighting on all lights or use the Specular Global pass.

The Specular pass omits the Diffuse light, Diffuse color, and reflection color influence. I disable shadows and raytracing before rendering. Note that reflections are added to the Specular global pass unless raytracing is disabled from the render global raytracing tab:

One of two ways to accomplish this render:

1 Select Emit Specular only for the lights, as well as disable raytracing.

2 Enable the Specular pass from the global passes; disable raytracing.

I render this pass twice, once as a regular Specular pass, providing the values for the Specular color and intensity. I use the other for the Specular bloom, which is the Specular glow of the "hot spots." I control this glow from the shader glow node (in the material sections of the hypershade window).

BIT DEPTH AND RENDER CHANNELS

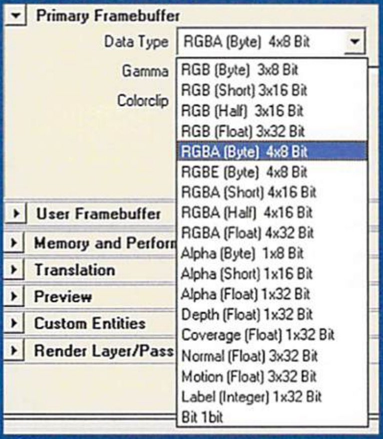
Before we get into rendering out layers with mental ray, let's examine the image output channels and bit depths of the output files. When rendering with mental ray, within the Common render tab, under Image File Output, the RGB and Alpha channel options are grayed out. mental ray provides control for these options within the mental ray global tab.

One advantage of rendering passes with mental ray lies within its ability to specifically set the render channels (RGB, A or Z), data types, and bit depth settings for the output files. It is likely that you need images with varying bit depth and different channels for compositing. A typical example is rendering Color passes at 16 Bit, providing for high quality images and rendering the Z-depth at 32 Bit, for better Z-depth integration while compositing.

The Primary Framebuffer section **(figure 1)** under the mental ray globals is dedicated to controlling image channels, bit depth, color clipping, premultiplication, and data types of your rendered image file. To select the image file type and bit depth in mental ray, you will have to make this selection in the Framebuffer Data Type option **(figure 1)**.

The Data Type dropdown list shows us the render channels (RGBAZ), followed by the data type in parentheses (variable type used to dictate the precision of the stored color value), and finally the bit depth per channel. So selecting "RGBA (float) 4x32 Bit" will render

Figure 1 – Bit depth



Primary Framebuffer

Data Type: RGBA (Byte) 4x8 Bit

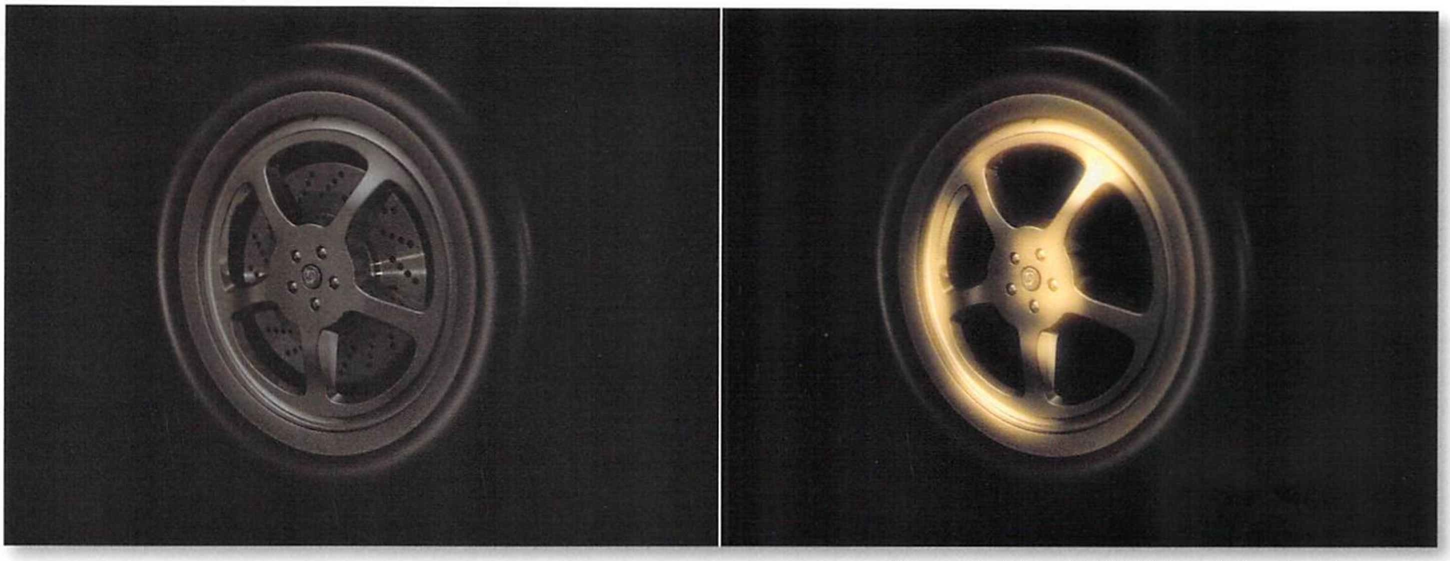


Figure 10 – Specular Highlight and Specular Glow passes

Reflection Pass **-** Retrieves the values for the reflection contribution

Framebuffer type: RGBA 8 or 16 Bit.

Method: Create custom shaders based on your needs.

RENDERING REFLECTIONS OF SCENE OBJECTS:

1 The best way to render a Reflection pass with Maya when you need reflections between objects is by using the Use Background shader. This shader renders reflections without any other color contribution, meaning only the reflection color is rendered, as well as an appropriate Alpha channel for the reflection value. Use the Use Background shader in a similar way for retrieving shadows (see Shadow pass). Disable Primary Visibility for all other objects (reflection casting objects) so they don't appear in the render, thus they are only visible in the reflections captured by the Use Background shader. Obviously this can get tedious if you need to extract reflections from several different objects. Set the Use Background shader Reflectivity value to one, Shadow Mask value to zero, white Specular Color, and specify enough reflection rays for the Reflection Limit attribute. It is important you don't have a Shadow Mask value, since the shadows would also appear combined with reflection values, unless you manually turn off Reflectivity on all objects.

2 Fake it – an old technique for reducing the render overhead by eliminating the need to raytrace to get reflections of scene objects. First, render an image using a camera angled from the reflection perspective. Map the image onto the reflective object's Shader Reflection Color attribute, or onto a surface shader Out Color value, as both will render color without the need for light or raytracing, thus transferring only the "visible" reflection value into the rendered image. A small experiment may help understand this technique.

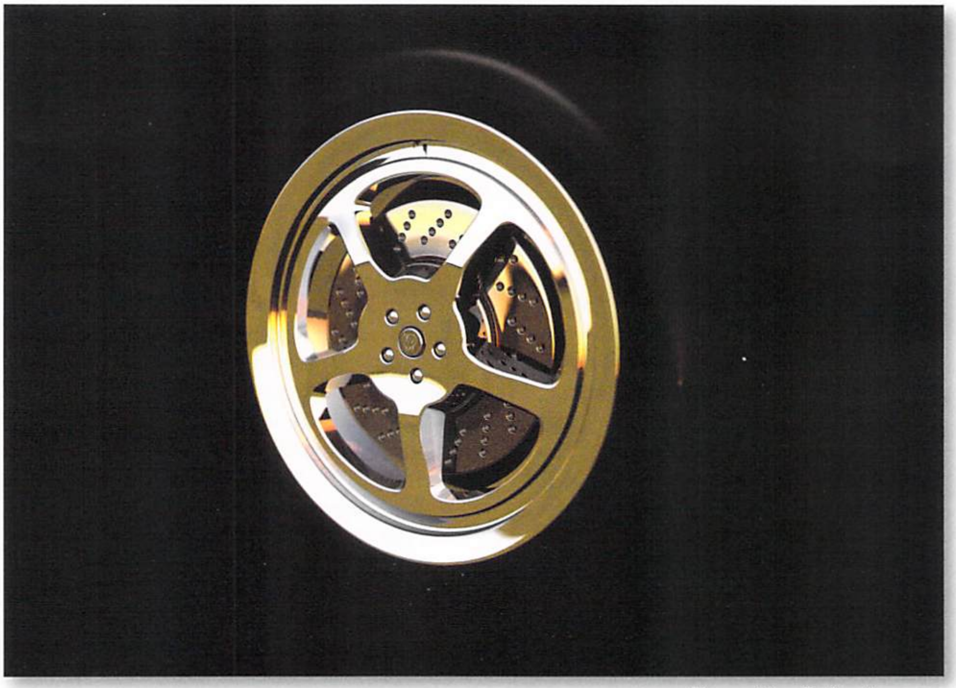


Figure 11 – Reflection pass

RENDERING ENVIRONMENT REFLECTIONS:

There are two primary ways of rendering solely environmental reflections. One is simply mapping the environment shader onto the Reflection Color or the surface shader Out Color. The other way is to retrieve reflections from an environmental dome in the scene such as an Image Based Lighting (IBL) node. For this you can use either the Use Background shader or a fully reflective shading model; with either render without any lights and with raytracing enabled. If you're unfamiliar with the IBL node, it comes from the IBL heading under the mental ray tab in the Render Global Settings Window. An IBL node is much like a mapped sphere that encompasses the scene, but has two very distinctive features; the IBL node acts as a light shader and is not rendered as a geometric object. IBL is used in several ways with mental ray and is beyond the scope of this article; however, we might examine it further in a future article. For more information on HDRI and image based lighting, refer to Dr. Paul Debevec's Web site, ***http://www.debevec.org.***

Note that for all the Reflection passes mentioned above, you should not render reflections with Specular Highlights, which would defeat the purpose of rendering separate Specular and Reflection passes. For this reason, we must avoid using lights that emit Specular light, including disabling the default light. When there is no Specular light left in the scene, there will only be reflections, even with a Specular shader like Blinn.

Follow these steps for rendering a Reflection pass with a Specular shading model:

1. Create a Specular shader, for example a Blinn shader using a black Color value, zero Diffuse value, white Specular color and fully reflective.

2. Disable or delete all the lights from the scene, and disable the Default Light in Render Globals.

3. Enable raytracing and render the scene.

This process results in a fully reflective pass, giving 100% environmental reflection across the surfaces rendered to be comped in. Use the Specular and Reflective Occlusion passes to scale the reflection intensity value across the surface as you composite the scene. *(Read more about Reflection and Occlusion Reflection passes in article 1 in this series, Issue #04, Ambient Occlusion with mental ray.)*

Shadow Pass –

Method: Enable the Shadow pass in the Render Global tab and remove all the shaders.

Framebuffer type: RGBA 8 or 16 Bit

Maya renders the shadow into the Alpha when using the Background shader or the Shadow pass from the Global passes. Invert the shadow color, which is white, during composting.

We retrieve the shadow information using the background shader, which renders the shadow details across a surface into the Alpha, without also rendering the surface geometry into the Alpha channel. This enables us to composite the shadows with live action. For example, rendering a shadow on a wall in a live action plate: Use a stand-in plane (modeled based on the wall) and assign it the background shader. Turn visibility off for the other objects in the scene, and they will cast their shadows onto the plane without being seen in the render. When rendered, the plane renders with only the shadow detail embedded within the Alpha, enabling us to transfer the shadow from the scene onto the live action plate.

Another way for retrieving the shadow for the scene is by enabling the Shadow Global pass. The shadow will also render into the Alpha channel, which means that the image appears to be black when viewing the RGB channels.

CUSTOM PASSES:

Alpha / Matte Pass – Masking objects.

Framebuffer type: RGBA 16 Bit.

In this case, 8 Bit would suffice, but if there were gradient transparencies in the alpha, I would rather use 16 Bit.

Method: - A few options available.

Optimize the render disabling raytracing and shadows, as well as customizing the sampling threshold to obtain a clean alpha.

1. Render the Diffuse Color pass with an Alpha in your Diffuse Color pass. You can then extract the Alpha during compositing from your Diffuse pass. Make sure the Contrast Threshold values match for the RGB and Alpha channel.

2. Apply a Surface shader with a white color to everything and render. This provides a Matte pass in the RGB channels.

3. If you need to further optimize the render, use low quality settings for the RGB channels and high quality settings for the Alpha channel. Do this by setting the lower values for the Alpha Contrast Threshold value. Be sure to only use the Alpha channel for compositing, and the RGB channels for viewing. Rendering a matter this way may be necessary if your compositor is not able to easily display the Alpha channel of an image.

Object separation:

Use the same Framebuffer and methods from above (Alpha/Matte) with a few additions.

Object separation is exactly like rendering with the Render Layers option enabled, only rather then rendering each object individually, I use masks or color coded images for creating separation within the compositing stage.

My goal is to divide and render the scene into separate objects. In this case, I separated the outer rim from the brake section; therefore, I can fine-tune each section individually within any of the passes. For example, I can tune the Specularity for both sections, independently.

For creating these passes I do one of the following:

1. Render the objects separately, by hiding other sections in the scene, using the Alpha/Matte method, and creating two Alpha passes, as seen in **figure 12**, or

2. Apply surface shaders to the objects. Each shader should have a unique color value that is easy to key within a compositing package, enabling you to select the objects you want to influence by keying out the

BIT DEPTH AND RENDER CHANNELS CONTINUED

four channels (RGBA), each using a floating point variable with a 32 Bit depth per channel.

By default, mental ray will render an .iff image with 8 Bit depth, unless you specify a higher range bit depth. Iff can be rendered out with most bit depths and data types, so is a good choice for most, if not all, tasks. For now, we can leave the data type and bit depth to RGBA 4x8 Bit (currently selected in **figure 1**). This provides us with all three color channels, as well as an Alpha channel, all at an adequate bit depth per channel.

Premultiply and Colorclip **(figure 2)** control the handling of transparency and color clipping before being their transfer to the Framebuffer. The Framebuffer ultimately is responsible for receiving color values for the rendered channels, and outputting them as an image file of the selected format and bit depth. Technically, the Framebuffer is a temporary image, stored in memory with RGB, A, and Z-channel values for each pixel. While rendering, the renderer transfers the color values for each pixel to the Framebuffer, which temporally stores them until the render is complete. Once the render calculates the color values for each pixel, any post render effects calculate for the temporary image stored within the Framebuffer. Eventually the render process is complete, and the Framebuffer transfers the pixel color values for each channel to the image file.

3D color values – especially with the introduction of HDR images – have a large dynamic range, so when rendering an image, it is very likely that the color values for that scene will actually exceed the standard 0-1 range of an image, giving us ultra white colors (HSV – V is greater than 1), as well as negative values of ultra-blacks. To control the evaluation of colors for the Framebuffer, we must provide colors within the range of the selected data type (8 Bit, 16 Bit...).

The Colorclip and Premultiply options control how the color values calculated by the renderer transfer to the Framebuffer. The Framebuffer supports values that are within the range of the selected data type and bit depth, thus without setting the Colorclip and Premultiply options correctly, it may receive color values outside the range that may result in

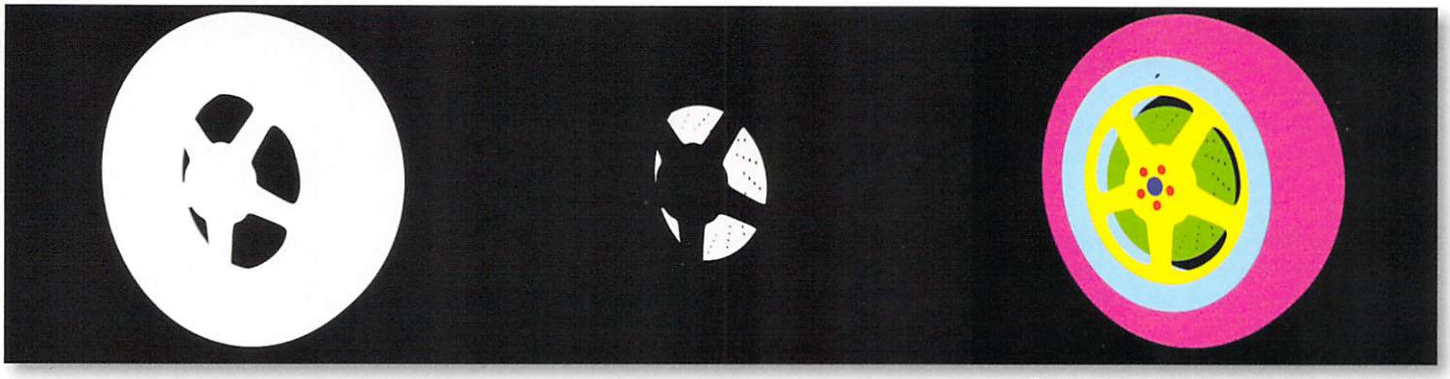


Figure 12 – Alpha passes and color-coding.

others. This acts as a color-coding method for objects, visible in **figure 12**.

Y-gradient pass **–** Enables control over the Y (up) axis in compositing, regardless of the perspective.

Framebuffer type: RGBA 16 Bit – for a higher resolution color transition.

Method: Create a planar projection in your scene that covers the range of your objects over the Y-axis, using a black to white ramp. This render layer is used for compositing special effects and color corrections; for example, rendering mist or fog influence at different heights.

Z-depth

Methods: mental ray Z-depth render or custom shader with a ramp.

Framebuffer type: Depth (float) 1x32 Bit or RGBA 32 Bit.

Z-depth is the process of rendering an image describing the distance from the camera into the scene (scene depth), resulting in a white to black gradient transition. It is common to have render artifacts appear in the image if not rendered to a high color range, so you should always use a 32 Bit image for Z-depth renders. Just select the Depth 32 Bit data type from the Framebuffer tab in the mental ray render globals. Z-depth is commonly rendered and visible in the Z-channel, such as with an .iff image file. You can also render Z-depth by enabling it from the Common render tab, under the image quality section, in the same way you would for the Maya software renderer.

The Z-depth channel is viewable using mental ray's image display utility (type imf_disp into the command line) and the Fcheck utility. Press the Z key with Fcheck to see the Z-channel. There is no easy way that I personally know of to view the Z-channel on a Mac using Fcheck or any other program; if you know an easy workaround, I'd love to know.

Before compositing using Z-depth, it is useful to convert the Z-channel into RGB color space, both for compositing and viewing reasons. Compositing packages that support Z-channels don't work well with the Z-channel anyway.

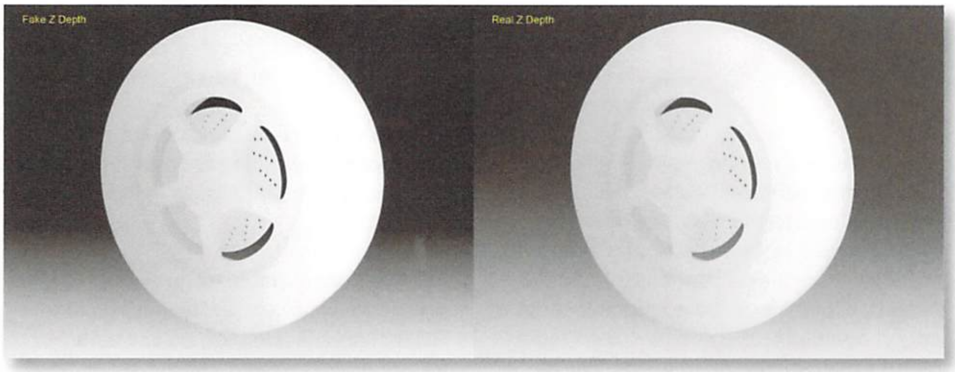


Figure 13 – Fake and real Z-depth passes

A way to get around rendering to a Z-channel is to render it into RGB space by using a custom shader that creates a fake Z-depth. One way for accomplishing this is as follows:

1. Create a plane in the top viewport that encompasses the entire scene.

2. Parent the plane to the camera and zero out its rotation so that it aligns with the camera.

3. Apply a Surface Shader with a 3D planar projection using a white to black ramp connected to the Out Color Value of the shader, similar to the Y-gradient render.

4. Parent the projection to the plane so it inherits camera movements.

5. Apply the surface shader to all the objects in the scene.

6. Hide the plane from appearing in renders (it is just there to ease the setup process)

7. Under the Framebuffer tab, set the Data Type to RGBA (float) 32 Bit.

This provides for a fake Z-depth that renders into RGB color space, controlled by adjusting the size of the plane.

Figure 13 displays a faked Z-depth image beside a real Z-depth image. Both passes rendered at 32 Bit depth, however for print convert to 8 Bit.

LIGHT PASSES:

Key Light, Fill, and Rim Lights

Framebuffer type: RGBA 16 Bit

These passes contain lighting information on a per light basis. The main scene divides into separate light categories, such as key lighting and fill lighting. The Key Light pass **(figure 14)**, contains only light emitting from the key light, enabling us to adjust the light intensity, falloff, and color during compositing.

The fill and rim lights can each be a separate pass, rendered in the same way. I will use the environment sampling pass as my Fill Light pass. More detail in that section.

For rendering the Key Light pass:

1. Assign a single shader to the entire scene, Lambert with a Diffuse value of 1, and a white Color value. This creates a grayscale render, describing the light falloff from white to black, over the scene. Adjust color and intensity further in composite.

2. Disable Shadows and R. In this case, it is better not to render shadows, as there is a separate Shadow pass.

Environment sampling **–** Retrieve the light influence from the environment

Framebuffer type: RGBA 16 Bit

Method: Occlusion texture or Final Gather

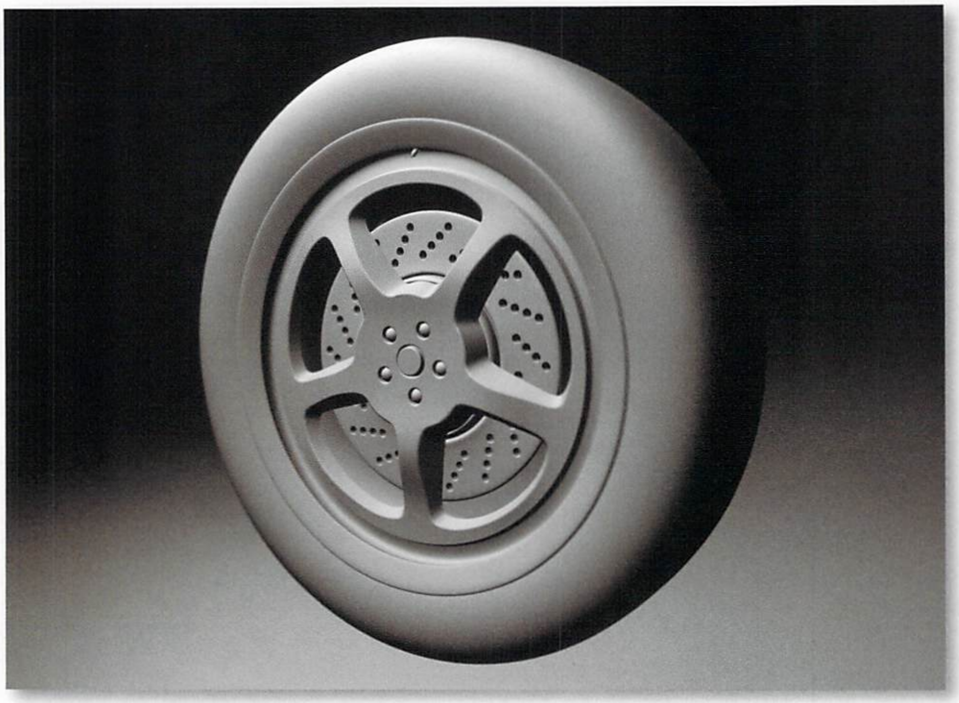


Figure 14 – Key Light pass

The Environment Sampling pass retrieves light influence from an environment; i.e. a scene in a red room creates an influence of red in your scene. For sampling the environment, I use an HDR image mapped to an IBL node (IBL tab in the mental ray Globals). mental ray samples the color intensities from the HDR image and uses them to describe the environment influence across the geometry surface. There is a Diffusive pass and a Reflective pass to pick up general tone and reflectivity, respectively.

In a nutshell, objects' colors are defined by the way light scatters off of their surfaces; or, in other words, the way light reflects from their Diffuse and their Specular components. The Diffuse component scatters light rays in all directions and is familiar to us as the Diffuse color. The Specular component scatters light rays in a mirror-like behavior, representing the reflection color, also known as the Specular color.

Both passes provide for the environment influence over the scene; the choice of which one to use, or both, should be an artistic one. Using both allows the scaling down of the reflective version, by the Specular and Reflective Occlusion passes, thus showing this Environment pass color in just the Specular highlights.

We will examine all these passes more closely in the next article during the compositing process.

To render this pass (using Maya 6.5):

1. Create a surface shader and apply to all the geometry in the scene.

2. Apply a mental ray Occlusion texture to the surface shader's out color value. This texture node is under the "Texture" heading in Create mental ray Nodes in the Hypershade in Maya 6.5 called "mib_amb_Occlusion." See the first tutorial in this series for more detail (*HDRI 3D* Issue #4) on Occlusion rendering, as well as rendering Occlusion with Maya 6.0 and earlier, since Maya 6.5 is the first version of Maya to build in the mental ray Occlusion Texture node. Set the Occlusion Texture Mode to a value of 1 (environment sampling).

3. Set Samples to 32 for testing, though you may want higher sampling for the render.

4. Create an IBL node from the mental ray Render Global tab

5. Assign an environment image to the IBL node (recommend using an HDRI)

6. Render the Diffuse sampling, and then enable Reflective and render the reflection sampling.

You can also use Final Gather for rendering the Diffusive pass for the environment:

1. Create an IBL node, using the Environment texture, same as above.

2. Create a Lambert shader with a Diffuse value of one and a white Color.

3. Enable Final Gather, and adjust the quality settings for the scene.

4. Render the Diffuse pass.

BIT DEPTH AND RENDER CHANNELS CONTINUED

unwanted artifacts, strange color evaluations, as well as an aborted render.

Premultiply, when enabled (recommended), controls how transparency values are stored within the RGB channels. This will multiply each of the RGB color channels with the alpha value and thus will never exceed that value. For example, an RGB value of 0.9 multiplied by an Alpha value of 0.2 (R*A, G*A, B*A) will result in a 0.18 value for each channel; this means that the RGB value will never exceed 0.2 and transparency is preserved.

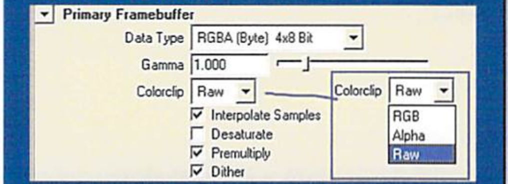
Colorclip controls how mental ray treats color channels with values that exceed the 0-1 range with respect to the selected image output bit depth, thus providing values suitable for the selected bit depth range. The reason for using Premultiplication with Colorclipping (RGB and Alpha, not RAW) is to scale the range of one value to the other, thus maintaining either the color or the Alpha channels value.

RGB – Will preserve the color values by clipping them within the range of the selected bit depth, then if premultiply is enabled the Alpha will be scaled to match the RGB value; this will preserve the color value and brightness.

Alpha – Clipping the Alpha preserves transparency within the range. If Premultiply is enabled, the Alpha has precedence over the RGB components, which will be scaled down to meet the Alpha value, thus clipping the color values to match the Alpha value and preserve transparency.

RAW – Enables colors outside the normal range and colors that contradict the alpha value. When enabled Premultiply and Colorclipping are off, this method should not be used with 16 Bit depth and lower.

Figure 2 – Colorclip



Primary Framebuffer

Data Type RGBA (Byte) 4x8 Bit

Gamma 1.000

Colorclip Raw

☑ Interpolate Samples

☐ Desaturate

☑ Premultiply

☑ Dither

Colorclip Raw

Ambient Diffusive Occlusion and Ambient Reflective Occlusion

Method: Occlusion texture or Final Gather
Framebuffer type: RGBA 16 Bit

These renders describe how occluding objects block light from illuminated points on the surface. The Reflective Occlusion pass provides an image that samples how the reflection value changes across the surface based on occluding objects or self-shadowing from tight corners and folds. These methods give us the ability to add great realism to images by controlling their natural self-shadowing affect, as well as the effect from other light blocking objects. For more information on these passes and their settings, please refer to "Ambient Occlusion with mental ray" from *HDRI 3D*, Issue #4.

Basically, follow the same steps as with the Environment Sampling passes, but with the following exceptions to render:

1 Set Mode to 0 in the Occlusion Texture node, which is the Occlusion Pass mode. Render the Diffusive pass.

2 Enable Reflective mode in that node and render the Reflective pass.

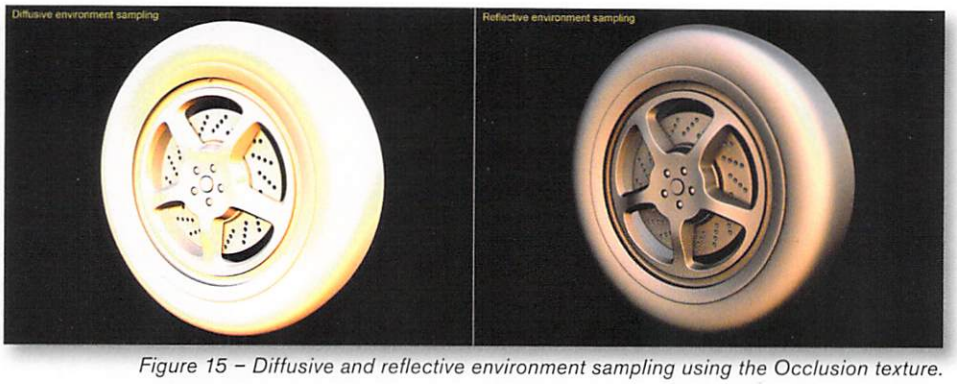

Diffusive environment sampling
Reflective environment sampling

Figure 15 – Diffusive and reflective environment sampling using the Occlusion texture.

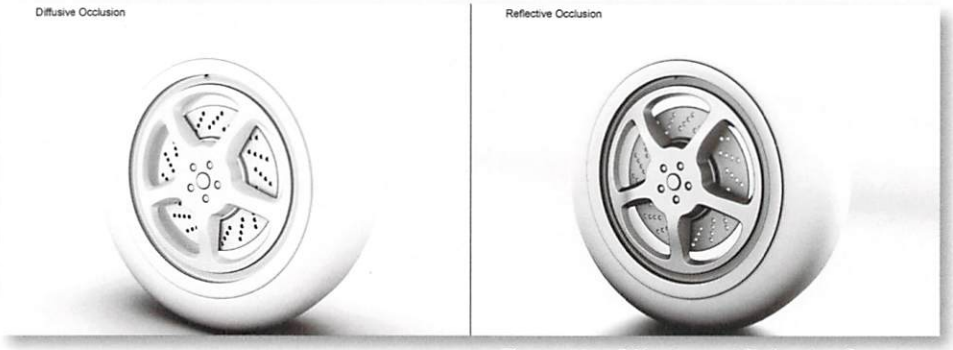


Figure 16 – Diffusive and Reflective Occlusion.

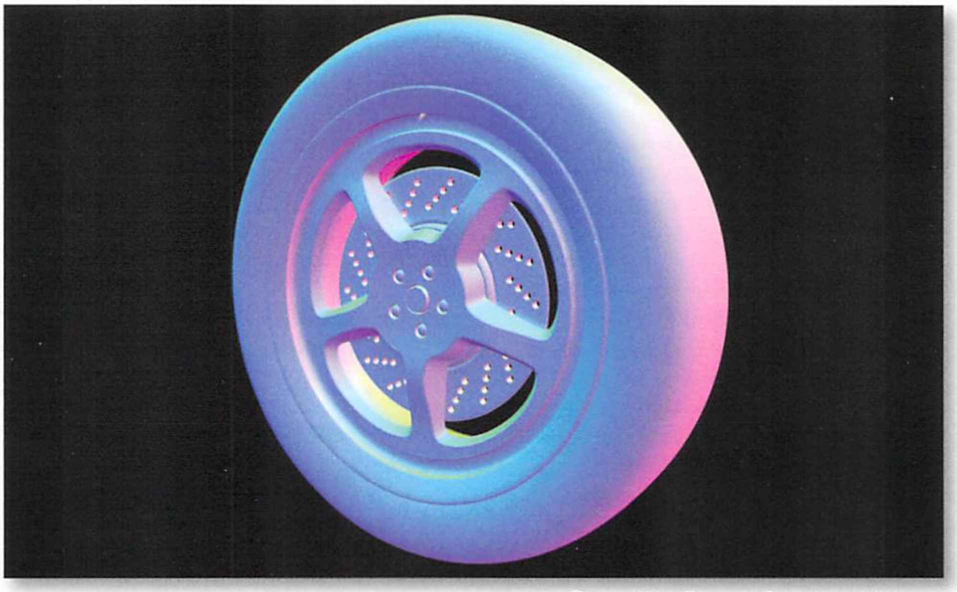


Figure 17 – Camera Bent Normals pass.

The Occlusion Texture provides better results than using Final Gather to render Diffusive Occlusion, as well as an ability to control Occlusion on a per object basis, using separate Occlusion textures. Final Gather calculates its solution based on a global setting, which may or may not work for the scene. For example, Final Gather may not work well when rendering objects with fine detail such as fruit in a bowl, as well as the Occlusion for the walls in the room.

For rendering with Final Gather, follow the same steps as with Diffusive environment sampling pass, with the following exceptions, use the IBL node, in Texture mode, with a white Texture Color.

Bent Normal – Retrieve the scene axis with color-coding.

Framebuffer type: RGBA 16 Bit
Method: Occlusion texture.

This render pass is exactly the same as rendering the Occlusion passes, using the Occlusion Texture. The only exception is setting Mode to 2 or 3, with no lighting or IBL nodes required. This pass is simply a Color pass. It provides separate colors for the X (red), Y (green) and Z (blue) axis, to give us more control during the compositing process, such as further adjusting the color treatment based on the camera axis and relation of the subjects to the camera. Mode 3 will provide bent normals in camera space, and Mode 2 provides world space coordinates.

IN CONCLUSION

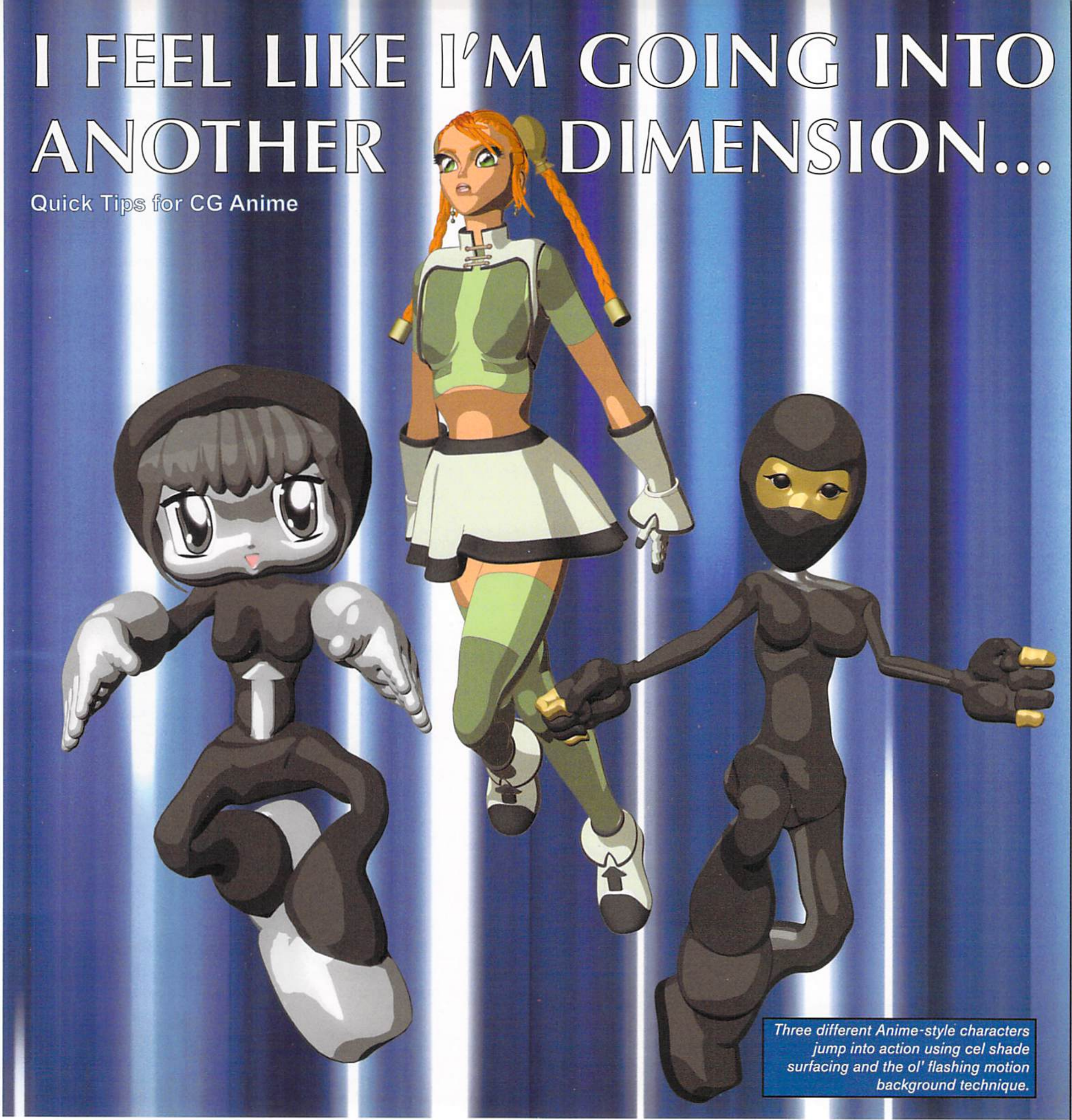
This article examined color evaluation and controlling the Framebuffer, as well as steps for achieving many different lighting passes. An understanding of color evaluation is critical for being able to recreate the render steps using passes with a compositing process, as are controlling your image channels and depth. This knowledge forms the foundation for approaching rendering passes and compositing at a fine level. The benefits lie in the ability to properly separate color with render passes, as well as retrieve custom passes that help achieve more control during compositing.

I hope you found this article educational. In the next article, I will use these passes with a node-based compositor, such as Shake or Nuke, for demonstrating the compositing stage and how these layers come together to form the image.

Boaz Livny *has been working with 3D for over 10 years, on film, TV, and multimedia content. He is a technically savvy artist that specializes in modeling, lighting, and rendering, with experience working the entire pipeline. Boaz's studio, www.visionanimations.com, is located in NYC and provides regular services to clients and freelance support for studios. He also teaches the Advanced Maya Course at NYU and is currently involved in establishing an advanced training center in NYC. He is also contributing to the next Sybex, Mastering Maya 7 book.*

I FEEL LIKE I'M GOING INTO ANOTHER DIMENSION...

Quick Tips for CG Anime



Three different Anime-style characters jump into action using cel shade surfacing and the ol' flashing motion background technique.

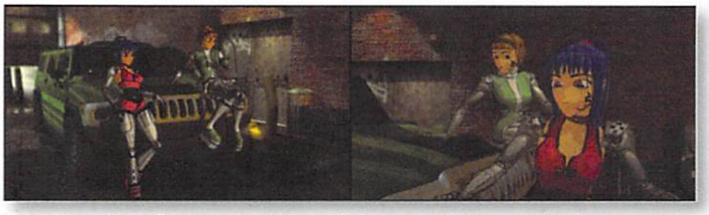
By William "Proton" Vaughan
3D Artist, Author, Teacher
Orlando, Florida

"I feel like I'm going into another dimension..... A dimension born of SPEED!"
-Speed Racer

With theater releases of *Steamboy*, *Appleseed*, and *Innocence* here in the states, it's no secret that Anime has finally reached the attention of your average moviegoer. Like a crack addict's first hit off the pipe, Akira sucked me into a life-long obsession with Japanese animation, always looking for my next fix. Once I started working in 3D, I quickly started to create Anime-style characters and produced cel shaded renders, trying to copy the look of traditional animation.

Recently, I wrote and directed a CG Anime-style movie titled *Spoonman*, a nine-minute animated short created in just under three months by the students at the Digital Animation and Visual Effects (DAVE) School. Through the three-month journey, we discovered a lot of tips and tricks that you may find useful if you decide to create your own CG Anime film. Here are 20 quick tips that should help get you started.

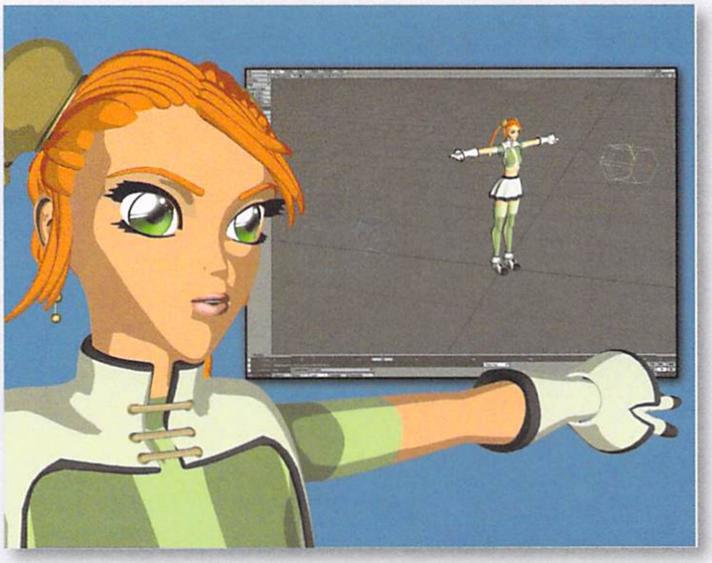
Tip 001



Two shots from Spoonman using cel shade surfacing on the characters, but not on the environment.

You don't have to cel shade every element in the scene when trying to create a 2D look to your animation or render. This is a common mistake made by many artists when they first start using cel shade techniques. If you look at traditional animation, the characters are painted cels and the backgrounds are usually detailed paintings that are more realistic. Try rendering your backgrounds in full 3D and your characters with cel shading, and then bring the two together to create a more traditional look to your animation.

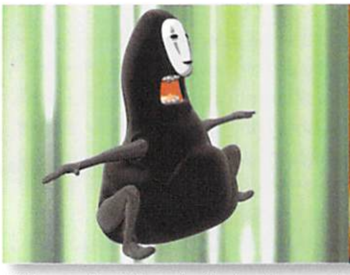
Tip 002



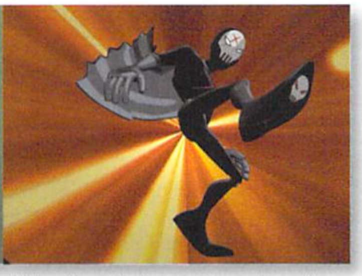
Using one Distant light to light this character was all that was needed to achieve great paint zones and give a traditional cel look.

Another good reason to render your backgrounds separate from your actors is to control the lighting for each element in your shot. Having too many lights for your cel shaded elements can really destroy the look of traditional cel characters. We found that one to two Distant lights used for lighting a character worked in 98% of our shots. Backgrounds, on the other hand, took full advantage of the various light types and amounts to give a more realistic look.

Tip003



Using vertical lines in the background gives the feeling that the character is moving vertically at intense speeds.



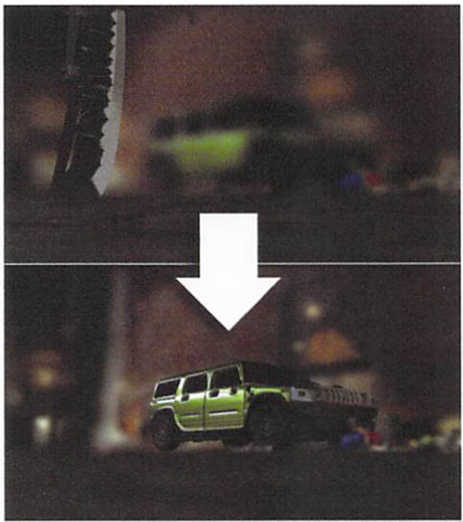
Using lines that move towards and away from the camera gives the feeling that the actor is moving in the Z-axis.

Sometimes, you don't even need backgrounds for a shot. Classic Anime action sequences often use flashing line backgrounds, which is a throwback to a technique used in printed Manga to how movement. This cuts down on rendering time and helps sell it as Anime, but be careful not to overuse this technique, as it can become the lens flare of Anime. Steve Primeau developed a quick technique using After Effects to create the backgrounds used in Spoonman and the images in this article. Be sure to check out his Web site at: **http://www.steveprimeau.com/projects/spoonman/index.html**

Tip 004

Most Anime is created in Japanese and then dubbed to English. If you plan on having your film shown in both languages, try animating to Japanese instead of English. This will follow the same production pipeline as a traditional Japanese animation, and could help match what Anime fans are used to hearing. We worked with ADV Films (www.advfilms.com) out of Houston, Texas to supply us with both English and Japanese voice tracks. Hardcore Anime fans will be very happy if they can watch your film in Japanese with English subtitles, and you open your film up to the Japanese market.

Tip 005



*When this shot from **Spoonman** starts the sword in the foreground is in focus, later the Hummer comes into focus where the action takes place.*

Pour on the rack focusing nice and thick, and don't stop until you feel you have overused it. You can't use the old rack focus technique too much in Anime. Using rack focus, the camera operator changes focus to shift viewer attention from one part of the scene to another. I lost count of how many shots in *Spoonman* have this technique applied to it. Check out any Anime and you will see them changing focus between foreground and background in many of their shots. When a shot just doesn't seem to have the Anime feel, play with the DOF and see if that does the trick.

Tip 006

Most Japanese animations have very small budgets, which forces them to get very creative with ways to cut production time and still tell a complete story. One technique employed quite often is to use environment shots with no characters in them and to have the actors speaking off screen. An example would be the scene in *Kite,* in which three people are in an elevator having a conversation, but for the majority of the scene, the only thing you see is the display showing that the elevator is traveling up. In *Spoonman,* we had an entire conversation take place off camera. This technique is a bit hard to swallow, but give it a try and you might be surprised at how well it works, and how much time you shave off of production.

Tip 007

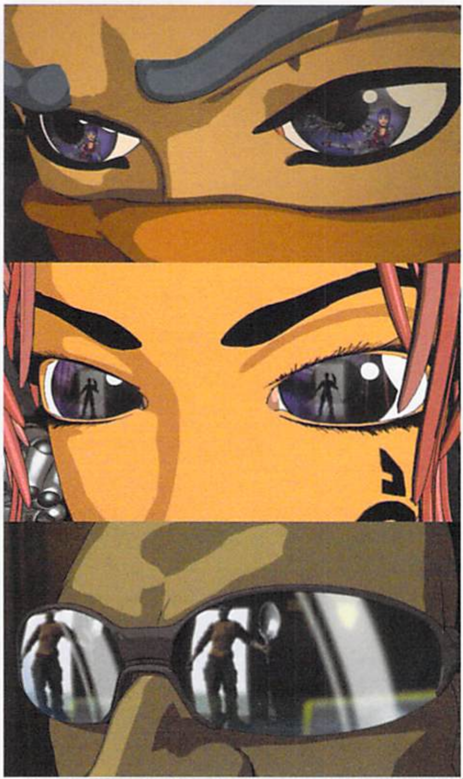
A traditional Anime technique that doesn't hold up in 3D, in my opinion, is freezing your actors while locking down the camera. This can save on production time, but we found that it is a bit jarring to watch, and we opted for minor animation on the actors so that the shots flowed much smoother. If your actors don't have any animation, simply do a slow push in with the camera. This is an example of how some techniques just don't cross over.

Tip 008

You don't need a lot of morph targets for Anime lip sync. Years ago, Human Code in Austin, Texas produced an Anime Trailer for an all-3D project titled *Avalon*. The characters in the trailer used only three mouth shapes: Open, Closed, and Open Wide. Although this was our plan for *Spoonman*, we found that it just didn't hold up for the scenes with heavy lip sync and we opted to use a few more. Using too many morphs also gave us mixed results, so we found the middle ground for each character.

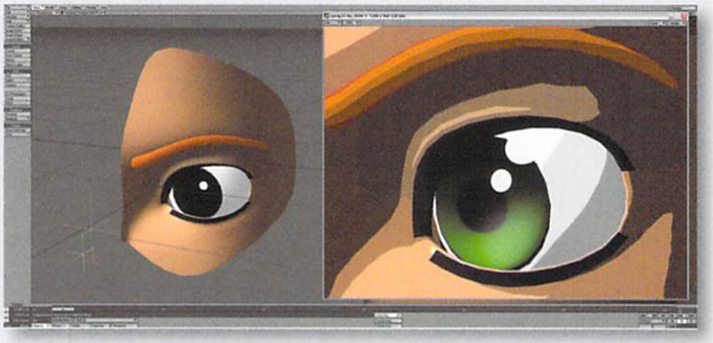
Tip 009

Close up shots of an actor's eyes is another common technique used in Anime. To really sell the shot, composite reflections of the other actor(s), or of the environment, into the eyes. Watch for an example of this technique in *The Matrix*, when Morpheus is talking with Neo.



*These three shots from **Spoonman** all use reflections as used in many traditional Anime films.*

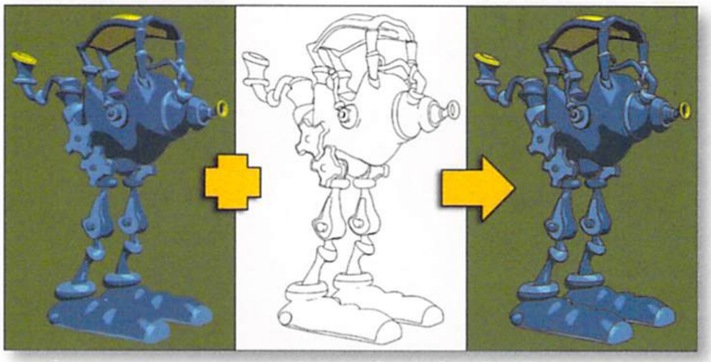
Tip 010



Using Specularity image maps referencing Null objects can be a quick way to animate the position and scale.

Another technique for close up shots of an actors face is to exaggerate the Specularity hits on the eyes. Controlling Spec hit using lights in your scene can be quite a challenge. Instead of using lights to control this, simply use image maps on the eyes. For intense scenes, randomly size the Spec hits over time to show emotion. Most traditional Anime films use this technique.

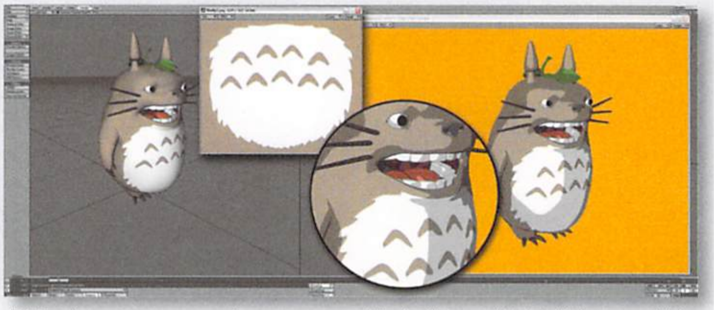
Tip 011



Two passes were rendered of this Crunk Car; Cel Shade pass and an Outline pass. These passes were then combined in post.

For full control over the actors' outlines, render the character with cel shade in one pass, and then the outlines of the character in another pass. This gives you the freedom to adjust each separately in a compositing program for ultimate control over the final look.

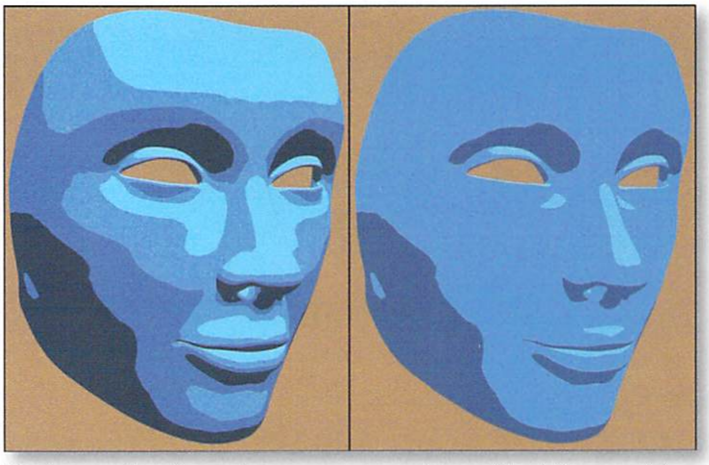
Tip 012



This character uses an image map on his belly to add detail that would be wasted geometry otherwise.

Don't be afraid to use image maps with cel shade. I would suggest limiting the images to flat color graphics so that they too look like they were cel painted. Image maps work great for patterns on clothes or tattoos on skin.

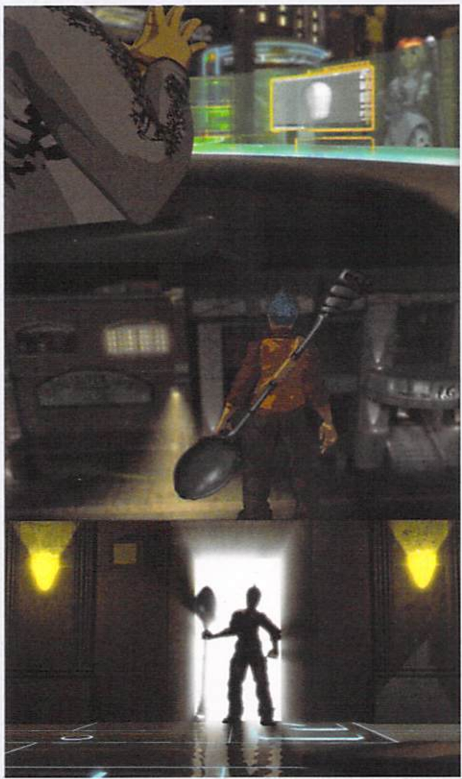
Tip 013



The face on the left uses too many zones and doesn't look like something a traditional cel painter would create. The Image on the right uses three zones and is much cleaner.

Most 3D cel shade work that I have seen uses too many zones of color. I'd suggest limiting yourself to two or three zones of color to match traditional Anime. Proper lighting and cel shade settings can affect the amount of zones.

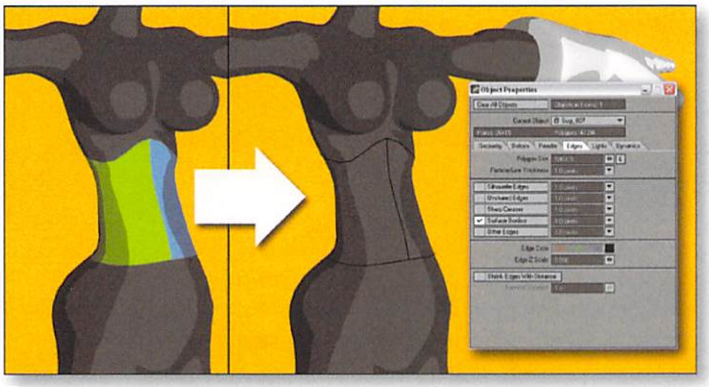
Tip 014



All three of these shots from ***Spoonman*** *use creative camera positions to avoid animating the face and lip sync.*

Want to shave time off of your production schedule? When an actor is talking, don't show his mouth. This is a common trick used heavily in Anime. Have the Camera over the shoulder of the actor speaking, or show the reaction of the other actors in the scene. This technique can knock massive amounts of time off your workload.

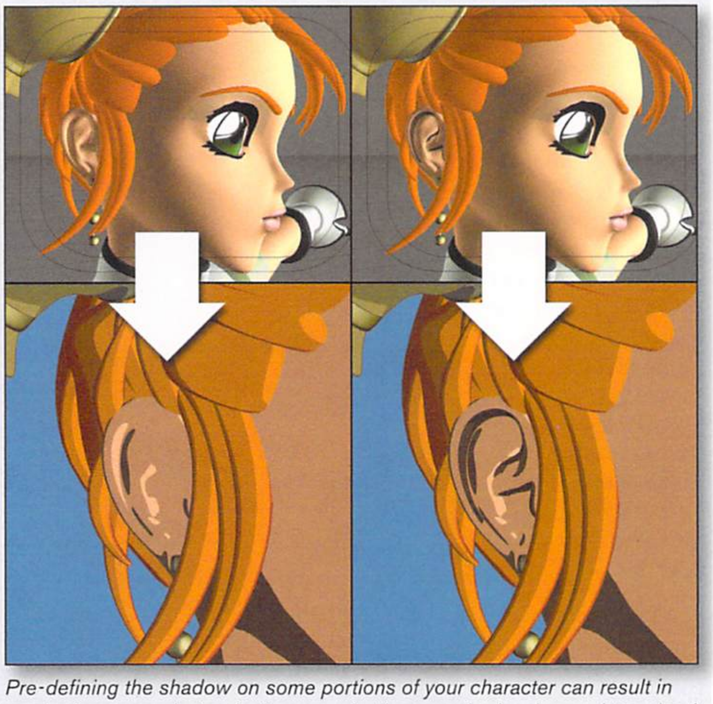
Tip 015



These three surfaces all share the same surface properties, but as separate surfaces, lines are drawn of the surface borders.

When working with outlines, sometimes you want a line to be drawn where it wouldn't normally by default using draw edges. One way to create a line is by splitting up the surfaces so that the line is drawn on the surface edges of two separate surfaces. Each surface may share the same surface properties and appear to be the same surface, but a line separates them. This is a great technique for patterns on a suit.

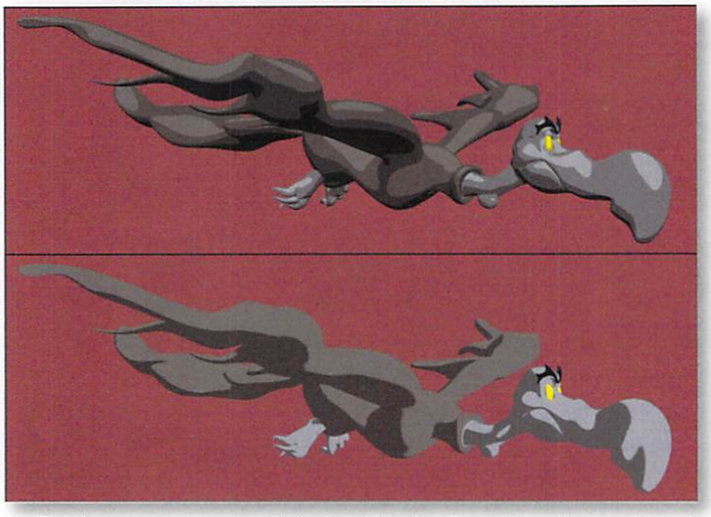
Tip 016



Pre-defining the shadow on some portions of your character can result in much cleaner rendering. In the example, the ears shadow is pre-determined by surfacing areas the color of the shadow.

Fake cel shaded shadows with surface color. A common problem area for cel shade is the ear. In some lighting conditions, the ear looks amazing until the light shifts or the character turns, then all the shadows disappear and you are left with a giant blob of one color. If you surface a portion of the ear the color of the shadow, you will be able to keep the detail shown in the ear at all times.

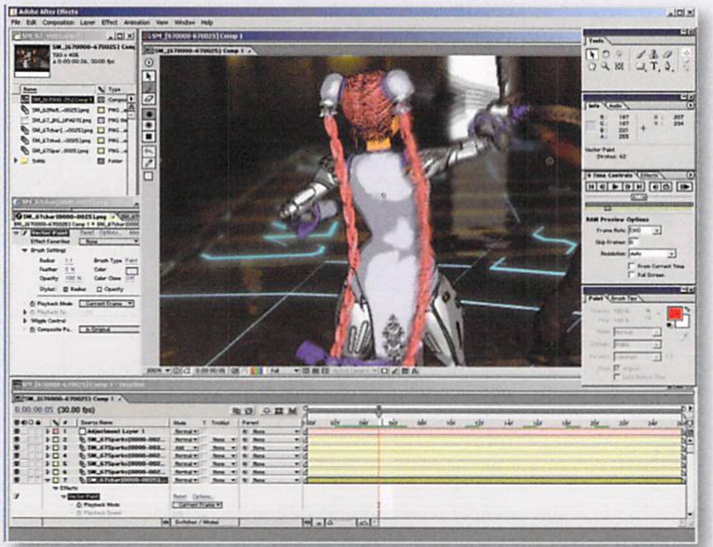
Tip 017



The bird on top uses the default cel shader. The bird on bottom doesn't use any shaders and the surface is created by changing Luminosity and Diffusion settings.

Don't be fooled into thinking you have to use the latest and greatest cel shader on the market. You can accomplish the same look, in many cases, with simple surface attribute changes. Try playing with the Luminosity and Diffusion settings and you might just surprise yourself at how close you can match a shader.

Tip 018



Using the Vector Paint tools in After Effects, Brian King made light work of refining the shadows on this shot.

Don't be afraid to do some post clean up on your cel shade work. On *Spoonman*, Vector Painting became just another step in the production pipeline. Instead of fighting the lighting or surface settings to fix a minor glitch in one of the cel shade zones, we opted to clean it up by hand in post, saving hours of headache and frustration.

Tip 019

If you plan on showing your film in Japan, pass the script by someone that is familiar with the Japanese culture. I gave the original script for *Spoonman* to Jun at Dstorm in Tokyo, and with a few changes, he helped fix areas that could have caused confusion to the Japanese audience. The biggest mistake that I see many artists making is using Japanese symbols without knowing their meaning. This could lead to some embarrassing moments.

Tip 020

As with most of the arts, Anime is evolving. Be open to try new things and experiment with blending techniques to develop a custom look and feel for your Anime-style film.

Anime is not a style everyone loves, but one thing is for sure – it's here to stay! It's influenced our animated series' on Saturday morning, our comic books, our live action films, and I hope that it influences your next animated film. Good luck with your CG Anime project, and I can't wait to see it in the near future.

Continue discovering CG-Anime online with these links:

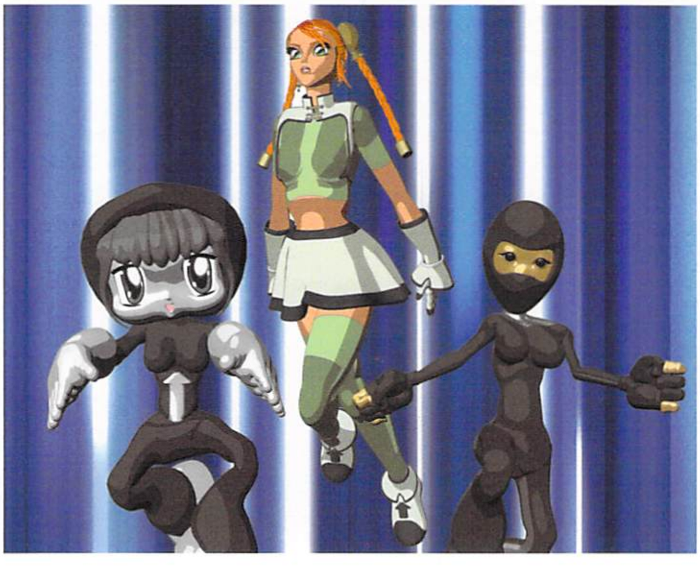
For up-to-date CG Anime tips and tricks, info, and more, be sure to visit ***www.celshade.com***.

Spoonman can be seen on the Dave School Web site, located here: ***http://www.daveschool.com/***

William "Proton" Vaughan *is a recipient of several New Media Addy awards. William has an extensive background in creative 3D for print, web, multimedia, games and broadcast. During the last 10 years, he has delivered award-winning work for clients such as Compaq, New Line Cinema and Halliburton. William has also trained artists at several studios and schools around the world and contributed to six LightWave 3D books throughout 2003 and 2004.*

In 2002, Vaughan joined NewTek's marketing team as the LightWave 3D Evangelist, working closely with the LightWave development team, key accounts, and the growing number of end users to enhance LightWave's features set.

William is Director of Industry Relations and Instructor of The DAVE School 's upcoming Final Project. William's focus is on continuously improving the quality of education at The DAVE School, while further establishing the school's presence in the industry.



UNCLE ROBIN'S MINI GUIDE TO MEL VOLUME 1:

The Way of Programming

BE NOT AFRAID

By Robin Scher
Animator, Programmer
Los Angeles, California

WHY DO I CARE?

The foundation of Maya is a sophisticated programming language called MEL. If you're just a cog in a monster industrial machine somewhere, with no motivation to improve your lot in life, then, yes, you may not ever have to use MEL directly. For everyone else, there will come a time when a half hour spent tinkering with a little script saves you days of work.

I experienced just such a time last week with a project I was working on, in which we got tons and tons of assets from another company. If you've never had this happen, consider yourself blessed, because as bad as our own organization skills may be, inheriting someone else's bad organization is a nightmare on a whole other level.

The particular scene files we inherited had been through a lot. They involved references and imports and all kinds of stuff that left me trying to sort through an Outliner with every node named something like:

Player_textured_FINAL:light_setup_hi_001:Hand

As you may have guessed, this made it rather difficult to find anything (since every node had the same huge and nasty prefix), and most of my monitor space filled up with these huge lists of text. Clearly, this wasn't helpful. But the scenes were huge, so there was no way I could go through and rename everything by hand.

And, lo! There is. MEL to the rescue. The first clue that you can use MEL is that the problem is very repetitive. Repetitive tasks are the ones best suited to scripting. The other clue is that the particular task involves a few discrete steps. This means that it is incredibly tedious to perform manually, but actually quite easy to write a script for.

There may be many different times you could use scripting, but this is the most obvious and is actually quite common. Remember the two clues:

- Same simple task
- Many items to perform the task upon.

ABOUT THIS DOCUMENT

Obviously, there are plenty of ways to learn how to script or program with MEL, and this article is only a brief introduction. It is meant as a starting point for Maya users who are relativebeginners to MEL, but it also has some hints for those that have played with MEL just a bit and run into some of the weird stuff. The tips and tricks in this article help you past those initial weird things and on to writing bigger and better scripts.

I have tried to use fonts and colors in order to help make things readable. **Blue** words signify MEL keywords or symbols, as you should type them yourself. **Violet** words signify commands. Keywords are actually part of the language, used to define or control execution, and commands give you programmatic control of the Maya tools themselves. **Green** is for comments, and **grey** for the contents of a string variable. I put code in a `monospace font.`

You may think it's cheating to put the fancy colors into my code samples, when you get nothing of the sort in the Script Editor, or the Expression Editor, or Notepad. Well, you may want to find a real code editor program. For Windows, I recommend Crimson Editor, which comes with built-in syntax highlighting for MEL. It's pretty cool, and I think it's actually free.

HELP!

By far, the most important thing about writing scripts or programs is where to get help. Believe it or not, even those of us that have been programming for decades still have to look things up almost every day.

There are two parts of the Maya documentation that you will refer to. The first is the section that actually describes the language; you will need it less the more you write scripts and get used to how it works.

The second, however, is the invaluable MEL Command Reference. Get to know this section, and know it well. Everything that you can do is here, and after so many revisions, it's actually getting very well written and understandable. Read it, know it, love it, it's your biggest friend. Well, second to this wonderful article, of course.

Finally, there is another great place to see how things work. Maya consists of tons of scripts, in the scripts directory under your Maya installation path. You should feel free to open scripts and see how the guys that wrote Maya make things work using MEL. Just be sure to make backups of them if you want to try changing things, or you may end up reinstalling Maya. Fun.

DEFENSIVE PROGRAMMING

If you ever write a single script, I guarantee you will make errors in it. It could be only two lines long, but you will make an error. Programmers, over a long time, have learned helpful tricks to keep errors out of code. These tips I call "Defensive Programming," and

I will explain them in detail in later sections. Here's a summary:

- Use shortcuts rarely.
- Write clean and pretty (i.e., readable) code
- Give your variables real names

SYNTAX

Every language has its grammar, and MEL is no exception. If you've ever used C or Java, the MEL syntax will look quite familiar to you. If not, no worries, it's not that complicated. There are a few keys that you need to remember.

Spaces

The space character is the most important character in the language, because it is what actually divides all elements. This is key, and will be confusing, because we also use spaces to divide up words in English, only differently. Plus, you will occasionally be mixing your English and MEL, so you need to be aware of where spaces are, and if you need to hide them somehow from the language.

In general, almost all MEL commands begin with the actual command name, followed by parameters, with everything separated by space characters. The parameters themselves should not have any spaces in them. But sometimes, you need to put a space in a parameter, and that's where quote marks come in.

Let's look at the **print** command, because this is one place where you will do this all the time. **print** displays messages to the user in the Script Editor. It is an incredibly useful command, because you can use it both for displaying progress messages to your users, and for your own debugging purposes. **print** takes a single argument as its parameter, the item to be printed. If you type this command into the Script Editor:

```
print hello there;
```

This is not going to work

You will get the error message

```
// Error: Invalid call to “print”. Check number and
types of arguments expected by the procedure. //
```

The result that Maya spits back out

In order to get Maya to do what you want, you need to make sure that it doesn't see that second space character. To do this, you use the quote marks. MEL considers everything inside the quotes a single entity. So when you type:

```
print “hello there\n”;
```

A string to display space characters

You actually see the result you intended. We'll come back to quote marks in a bit.

And the **\n** at the end of the string? Always put that in as is. These two characters signify the end of a line in the Script Editor's display, like a carriage return for those of you that remember typewriters. If you didn't use them, all of your **print** statements would end up on one line, and you wouldn't be able to read anything!

If you need to actually put a backslash into the display, use two backslashes in a row; these convert by Maya into just one when it displays the string. It gets really fun when you start trying to do things with UNC paths, believe me. When in doubt, try double backslashes if some variable with backslashes isn't working like you think it should.

Braces

If you only ever execute a single command, you will not actually need to use braces. But that's not very likely, so you should know what they are. Braces refer to the curly squiggly lines that look like this: {}. You will find these characters over on the right side of most American keyboards, usually by pressing shift and one of the square bracket keys (which we'll also see later, in the Variables section).

When you want to use more than one command in a row, you should surround the group of commands in curly braces, so MEL will consider it a single command block. You do not actually need to worry about the layout of the braces and you do not need to separate the braces with spaces. However, you will generally find that if you lay things out nicely, your code will be more readable, and easier to modify or fix later on. It's a good idea, then, to pick a layout style and stick with it.

The following code listings all produce identical results. You will find you make fewer basic syntax errors when you write your script if you write it cleanly and consistently.

```
if(frame==4){
   DoSomething($variable); DoMore($name);
}

if (frame == 4) {
   DoSomething ($variable);
   DoMore ($name);
   }

if( frame == 4 )
{
   DoSomething( $variable );
   DoMore( $name );
}
```

Different ways of saying the exact same thing

Quotes

As we saw before, we use double quotes to group together a bunch of words that you want MEL to consider as a single parameter. This is a string (think of stringing a whole bunch of characters together without caring what the characters are).

However, there is another type of quote you will be using all the time. This is the single back quote, a character you have probably never typed before. Generally, the key on English keyboards is in the upper left corner, and produces this character: `. You will need to use these quotes when you want to use the result of a command for something in MEL.

For example, you can use the ls command to list nodes in your Maya scene. Say you want to get an array of strings that is filled with the name of every transform node that begins with “ball.” You can do this using a command like the following:

```
string $allBalls[] = `ls -tr “ball*”`;
```

Example of single back quotes to access a command's results

Notice that the single back quote marks go outside the entire command, including all its parameters, even if double quote marks encase those parameters.

Don't try to nest quoted blocks together; it just won't work. If you need to have a double quote mark inside of a double quoted string, you need to “escape” the quote mark. This means, you put a backslash in front of it. For example:

```
print “Do you like \”quote\” marks?”;
```

Example of using “escaped” quote marks inside of a string

To nest single back quote blocks, use variables. Assign the result of a command to a variable, then use that variable as the parameter to the next command.

```
string $selection[] = `ls -sl`;
string $copy = `duplicate $selection[ 0 ]`;
```

Example of using the result of one command as a parameter for another

Semicolons

Ah, the semicolon. Most of us have no real idea how to use them correctly in English, but that doesn't seem to stop us. Unfortunately, you need to know how to use them correctly in MEL, or your scripts will never work.

Repeat after me: All commands must end in a semicolon. All commands must end in a semicolon. Any exceptions? No! *All commands must end in a semicolon.*

The semicolon signifies the end of a command. This is important. If you have two commands on a single line, separated by a semicolon, they count as two individual commands. If you have one command stretching across five lines, it's only one command. While these look completely different to you and me, to Maya they look exactly the same, and will produce an identical result. Which one would you rather see when you're reading lines of code?

```
// One long line can sometimes be hard to read
sphere -p 0 0 0 -ax 0 1 0 -ssw 0 -esw 360 -r 1 -d
3 -ut 1 -tol 0.01 -s 8 -nsp 4 -ch 1;

// Breaking it up can make it more pleasant to read
// but it still has the exact same effect
sphere
    -p 0 0 0    // pivot at origin
    -ax 0 1 0   // Y axis
    -ssw 0      // start sweep at 0
    -esw 360    // end sweep at 360
    -r 1        // radius 1
    -d 3        // cubic surface
    -ut 1       // use tolerance
    -tol 0.01   // tolerance of 0.01
    -s 8        // 8 sections in sweep direction
    -nsp 4      // 4 spans in non-sweep direction
    -ch 1;      // turn on construction history
```

Breaking up lines can make it much easier to figure out what's going on

Now, of course, Maya has made some shortcuts that allow you not to enter the last semicolon in a few circumstances. All I can say is, ignore these shortcuts and always put a semicolon at the end of your command.

All commands must end in a semicolon!

Now that you know there are no exceptions to using semicolons, there are a couple exceptions to watch out for. Some keywords are not actually commands, and thus should not actually have semicolons after them. For example, you don't put a semicolon after a **for** statement, because you instead put some command or command block that you are executing in a loop with the **for**.

Comments

Writing that cool script, you know everything that's going on, and have no problem remembering why you did things a certain way, or what a variable is used for later on, and so forth. But, I'm sure you've all experienced that time when you have to go back to something you haven't looked at for six months and figure it all out again.

With your Maya scenes, you have the ability to name your nodes. This is like giving your script's variables real names that correspond to their function, and that makes it easier to figure out what the variable does. You also have the ability to attach comments to your node through the Attribute Editor, leaving yourself instructions for later. MEL has something similar to keep your work organized, and the term for it is "comments" in your code.

There are two types of comments in MEL. Type one:

```
/* This is a comment */
```

An example of a comment

Everything inside of the slash/asterisk combinations is a comment; Maya completely ignores it during the execution of your script. These comments are handy, because they can appear anywhere, and everything between the **/*** and the ***/** will simply be ignored. You can even have line breaks in there, so these are great when you want to temporarily disable some section of your script, say while you're debugging it. Just put a **/*** in front of the section you want to disable and a ***/** at the end, even if it's a few lines down the script. Remember, absolutely everything between them will be ignored, so if you have some flow control keywords in there, make sure that the flow will go as expected around your comment. Otherwise, this can bite you in the butt.

Type two:

```
// this is another comment
```

Another example of a comment

The other comment type is even easier to use. If you put two slashes together, Maya ignores everything from those two slashes to the end of the line during the execution. This type of comment is extremely useful for little one line hints about a line of code or the use of a variable. They are simpler to type, because you don't need to worry about the end of the comment. See the broken line example above.

When you come back to that in six months, you will have a much easier time when you have commented your script. Rule of thumb: comment everything, even if it's just a quickie little script. It really doesn't take much time, and it will save you tons of time later on. I guarantee it.

Indentation and whitespace

Indentation and whitespace (i.e., spaces, tabs, and carriage returns) are valuable tools for you as the programmer or script writer, because they have almost no meaning to the program's execution. That means, you can use them to your advantage as the author of the program. We've already seen some cases of this, with the comments and braces examples.

Remember, it's just as important for your code to be human readable as it is for it to be machine readable. You're the one that has to work with it, and understand it, and it's even more important if you have to share the code with other people, because people (thankfully) are not computers.

COUNTING

Yes, you think you've known how to count since you were 2, right? Well, guess what, you're not counting right for a computer. When dealing with computers, always start counting from zero. It will be incredibly difficult at first, and will leave you confused a lot. But, over time, you will start to see that it actually makes your life easier, and reduces the number of times you will need to add or subtract one from things, especially when doing some kind of looping over an array of items.

An example of this would be with emitting particles. The first particle emitted is 0 and not 1, so when its particleId needs to be called out, it needs to be called out as 0 and not 1. As far as code, the first number in an array would also be in position 0 of the array, and not 1.

VARIABLES

Sometime late in middle school, between praying for your acne to go away and for the hot girl to notice you, you probably went to an algebra class. Here is where you first learned about variables and when you started seeing letters in math equations.

A variable is simply a named placeholder for a particular value. In MEL, it's super easy to figure out which words refer to variables, because all variables must start with the dollar sign ($) called a string.

Variables come in different types, even if you don't assign it yourself. Variable types are:

`int`	An integer value (1, 2, 3, 4, etc...)
`float`	A "floating point" or fractional number (3.1415, .5, etc...)
`vector`	A triplet of floating point values (<< 1.3, .4, 6 >>)
`bool`	A Boolean value, that can be either **true** or **false**, and nothing else
`string`	A sequence of characters ("`Hello there!`")
`matrix`	A two dimensional array of floating point values

Because a variable always has a type, it's a really good idea to specify the type yourself, just so you're sure. It's one less place for errors to creep into your code, because if you try to assign something of the wrong type to your variable, MEL may be able to notify you ahead of time:

```
string $foo = `ls`;

// Error: Cannot convert data of type string[] to
type string. //
```

Result of trying to assign the wrong type to a variable

In this case, the **ls** command returns an array of strings, so assigning this array to a single string results in an error. If you had just typed:

```
$foo = `ls`;
```

It is dangerous to let Maya determine the type for you.

Maya would have implicitly created $foo to be an array of strings. This may work fine, but it may also cause your script to fail, if you are using $foo as if it were of a different type somewhere else.

Always give your variable a real name that corresponds to its function. When you're reading through your script, it's a lot easier to know what a variable named **$windVelocity** means than if you had named it **$wv**. The extra half second of typing in the beginning will save you tons of time later, when you have to trace through your entire script to try to figure out what the hell "$wv" is supposed to mean.

Arrays of Variables

The type of your variable is almost always determined by the word you put in front of the variable name the first time you write it in your script. There is one exception, though. To create an array (which is a set of variables, like a laundry list of data), you need both the type in front to signify its type of data in each element of the array, and the two square bracket characters as a suffix, like this:

```
int $sizes[];
```

An example of declaring an array

This declaration says to MEL that you want to have a variable named $sizes, and that it is not just a single integer, but an array of integers. An array is like a list of items, where you can access any one of them by an index number. Remember, as I said above, that arrays start counting at index zero. So the second item in the array actually has the index number 1.

FLOW CONTROL

Flow control is programmer-speak for changing the order of the execution of your script's commands. Think, "controlling the flow of execution." It is a fundamental aspect of all scripting and programming languages, and consists of a few simple concepts that are pretty much the same in every language. Understand the concepts and your scripts will become much more sophisticated and useful.

One Way Street

Computers are stupid – and I'm not talking about things like spam and pop-up ads, which are entirely the fault of human beings. What I mean is that a computer just does exactly what it's told, one item at a time, in order, with no going back. A computer cannot guess what you meant to type, but can only do exactly what you tell it to do.

A code block is a logical grouping of some code that acts like a black box. Once Maya starts plugging down the commands in a code block, it executes every command in the block, in order, and the only thing that will stop it is an error in the execution of a command.

Now, a block of code can be as short as a single command, but it can also be much longer. This is where those curly braces come in. Remember what I said before: if you want to use more than one command in a row, surround the commands with curly braces. Maya considers everything inside of a set of braces a single code block, plus, you can spot it easily. This is really important when it comes to using the other flow control systems we talk about below.

There will be times when you just need a single command code block, and then you can leave off the braces. But more often than not, you later go back and add more commands to that block. So, my advice to you, when you're writing scripts, just use the braces for all of your code blocks, even if they're just a single command. This will greatly reduce the number of times you forget to put them in, and have a line of code executing at the wrong time, because it's not part of the flow control mechanism that you think it is.

Branching

Sometimes, you want to do different things depending on the existing conditions. We call this "branching." It's like hitting a fork in the road and determining which path to follow. Once you start down one path, the other is no longer available.

In Maya, there are two ways to branch. One is to use the **if** keyword, and the other is using the special character **?**. They each have their place, but for now, I would suggest sticking with **if**. It's a little more typing, but it's a lot easier to read, at least as you're still getting used to it.

The basic idea is that you have a little logical test going on, and the result of your test will determine which branch your program will follow. The actual syntax of **if** is:

if(*condition* **)** *true-code-block* [**else** *false-code-block*] construct starts with the word **if**, and is considered a single code block to Maya. This becomes important when you're trying to figure out which **if** block an **else** statement goes with. More on that in a second.

Your condition appears inside the parentheses after the **if**, and is evaluated until a definitive logical **true** or **false** is reached. If the result of that evaluation is **true**, then the *true-code-block* is executed. Remember, from the code block section, that this can be a single command, ending with a semicolon, or it can be a whole bunch of commands, surrounded by braces.

```
bool $true = true;
if( $bool )
    print "$bool is TRUE!\n";
```

if *example.*

You can optionally supply another code block that is executed if the *condition* evaluates to **false**. This block starts with the keyword **else**, and, again, can be either a single line of code ending with a semicolon, or a whole block of code surrounded by braces.

```
bool $true = true;
if( $bool )
    print "$bool is TRUE!\n";
else
    print "$bool is FALSE!\n";
```

if ... **else** *example.*

I can hear you asking – what the heck is a logical **true** or **false**? Well, one obvious possibility should be variables that are of the type **bool**, also known as Booleans. These are variables that have one of only two possible values, **true** or **false**, and are very useful for storing or calculating if something should or should not happen, and then basing a bunch of different calculations or commands on that.

But you're not limited to that; use "logical operators" on those. This is like math from high school now, with those stupid truth tables. I hated truth tables. Luckily, there's only maybe two you'll ever use, so you only need to know those. The logical operators are **!**, **&&** and **||**, which represent NOT, AND, and OR, respectively. In case you have no idea what I'm talking about, here are the truth tables for NOT, AND, and OR. The idea is that if you are comparing two Boolean values, look up the state of each variable, and look up on the table what the result of the calculation will be.

example	*inputs*	*result*
`!$value`	`$value is true`	`false`
	`$value is false`	`true`
`$value1 && $value2`	`$value1 is true and $value2 is true`	`true`
	`$value1 is true and $value2 is false`	`false`
	`$value1 is false and $value2 is true`	`false`
	`$value1 is false and $value2 is false`	`false`
`$value1 || $value2`	`$value1 is true and $value2 is true`	`true`
	`$value1 is true and $value2 is false`	`true`
	`$value1 is false and $value2 is true`	`true`
	`$value1 is false and $value2 is false`	`false`

The parameters for your logical tests are not limited to Boolean variables. Some types of expressions have a Boolean result, and you can use these in your condition. Commonly, these are more math terms, like == (is equal to), != (is not equal to), < (is less than), > (is greater than), <= (is less than or equal to), >= (is greater than or equal to). For these, you can use just about anything, from a constant number to a variable to the result of a function.

Be sure to notice the difference between a single equal sign, which is the assignment operator (x = y means set x to the value of y), and the double equal sign, which is the logical comparison operator (x == y means check if the values of x and y are the same). They are not interchangeable, and you will get unpredictable results if you use the wrong one. Unfortunately, their misuse does not always result in a syntax error that Maya reports before trying to execute a script. So, always pay close attention. If you want to assign a value to a variable, use the single equal sign. If you want to compare two values, use the double equal sign.

Finally, you can group your logical tests using parentheses. Maya evaluates the comparisons from the innermost to the outermost. If you leave off the parentheses and your **if** condition has more than one or two terms, you may be surprised by code not operating correctly due to the order in which the terms are evaluated.

```
bool $do_check;
bool $is_visible;
bool $enabled;

if( $do_check && ($is_visible || $enabled) ) {
        // to get here, $do_check must be true
        // and either $is_visisble or $enabled must
be true
        doSomething $object;
}
```

Example of using parentheses

You can get a lot of complexity in your **if** condition. Now, that's cool and all, but it's also error prone, so be careful, and try to avoid making them too complex.

```
float $position;      // some calculated position
bool $enabled;         // a switch from a user option
maybe?
string $object;        // the name of the object
we're messing with
int $index;            // some calculated index value

if( $enabled &&
    $position >= `getAttr ($object + ".tx")` &&
    (getIndex( $object ) == $index + 1 ||
    calcPosition( frame ) < .5 ) {
    // to get here, $enabled must be true,
    // $position is greater or equal to the
translate X of $object,
    // and either the getIndex proc for $object
returned 1 more than $index,
    // or the calcPosition proc for the current
frame returned a value
    // less than .5
    doSomething $object;
}
```

A really complicated **if** *condition*

As I said before, especially when you are new to MEL scripting, you should almost always surround your code blocks in braces, even if they are only one line long. **if...then** sections are a very good reason why. Check out this example.

```
if( $value > 5 )
   if( $value == 8 )
      doCommand 8;
else                    // Which if does this go with??
   doCommand 0;
```

Try not to do this

To which of the two **if**s does the **else** match up with? Notice that I have indented it to match the first one, but does it? Actually, remember that the entire thing is considered a single code block, so actually, that **else** would match up with the second **if**. Confused? You should be. The whole thing would be much easier to code, and much less prone to mistakes, if you would just use those braces:

```
if( $value > 5 ) {
   if( $value == 8 ) {
      doCommand 8;
   }
}
else {
   doCommand 0;
}
```

This is much clearer

No problems this time, because we've specifically defined the code blocks for the **if**s. Once again, a few extra seconds of work saves you hours of debugging when you're trying to figure out why doCommand was not getting executed when you expected.

Looping

Very often, you want to do the exact same operation to a whole bunch of things. Looping is the key to this sort of thing. Maya has several ways you can define loops, including **for**, **for-in**, **while**, and **do-while**. Each has its place, but I would strongly suggest sticking with **for-in** loops for most things.

The most common reason you will be using looping in MEL is to perform some set of operations on a whole bunch of objects. Remember our example at the beginning? We want to rename every node. The set of operations is to figure out what the name should be, then call the **rename** command to unleash the fury. So to speak.

Well, the **for-in** loop is devised just for this sort of thing. It allows you to loop through an array of items, one item at a time. Here's the syntax:

for(*element* **in** *array* **)** *code-block*

The *element* and the *array* are usually variables, and their base type must match. As you can probably tell from the names, the *element* is a single item of the type, and the *array* is an array of objects of the type. The *code-block* executes with the element set to each value in the array in turn. If there are five items in the array, the code block is executed five times, with the value of the element variable set to each of those five elements in turn.

An example would probably help clear it all up. We want to set up a **for-in** loop to check out every item in the scene to rename it, right? Check this out:

```
string $element;          // This is our element
variable
string $array[] = `ls`;    // This is our array we
want to loop through
for( $element in $array )
{
   // Get the new name
   string $newName = `substitute "Player _
textured _ FINAL:light _ setup _ hi _ 001:" $element
""`;
   // And rename it
   rename $element $newName;
}
```

For-in *example*

Here we have an array of strings that contains the name of every object in the scene. For each object, one at a time, we execute the **substitute** command, to get a new string of the name without the nasty prefix, then we execute the **rename** command to actually rename the object. Make sense?

It's all starting to come together now. We have our variables, we have our loop, we have our comments and whitespace, and we're actually doing something! Now, if you actually try to use this command as it exists, you will still run into problems, but let's hold off on that until the next article in the next issue....

You will, from time to time, have the opportunity to use the other looping systems. The other systems have some gotchas that you need to watch out for. First, make sure that you are actually changing the thing you're looping through with each loop iteration. In other words, if you had some array of numbers, make sure that you are actually changing the number you are using each time through the loop. Second, make sure that there is some way to stop looping. Otherwise, your loop will go on forever, or until you kill Maya and lose all your work.

If you stick with the **for-in** loop, these two things are taken care of for you.

Stopping

Sometimes, in a loop, you need to stop looping, say, if something went wrong, or the processing finished. You can use the **break** command to do this, and as soon as it's processed, execution of the command block will stop. The next statement executed will be the very first command outside of the program block, if there is one.

TO SAVE OR NOT TO SAVE

There are several different places in Maya where you could use MEL. Each has its particular differences, and the difference that you will probably notice most is if you have to save the file to disk or if you can just type the command in directly.

If the only time you ever use MEL is for expressions, you never have to worry about saving the MEL to a file and sourcing it, because the code is actually part of the Maya scene, and gets executed by Maya automatically when you're doing whatever triggers the expression, like stepping forward a frame or running up dynamics.

If you are just trying to automate some of your tasks, then you need to make the code accessible to you somehow as you're working in the scene. You have two ways to go here. Method one is to just drag and drop some code from the Script Editor onto your shelf. It shows up as a really ugly grey icon, and will automatically execute, starting with the first line you copied, and ending with the last you copied, no more, no less. This is handy mostly for little tidbits and shortcuts, or for starting a complex system that has its own UI.

Your other choice is to use a text file. Now, this is where things get kind of tricky. What Maya does with these is that it loads them once into memory, and then runs them directly from memory. This is much faster, but has the downside of having to force it to reload the script as you make modifications. You use the **source** command to load your modified script file before trying to use it. You can re-source a MEL file as often as you like (well, most of the time), but some types of changes won't be recognized until you actually restart Maya. For example, sometimes changing the number or type of parameters to a global proc you declare in your code makes Maya complain, and you need to restart it to get it to accept the change.

If you're the kind that likes to actually try to make Maya even cooler with the coolest control panel interface or even more powerful with the latest polygon editing tools, you should definitely just save the file and share it with the rest of us.

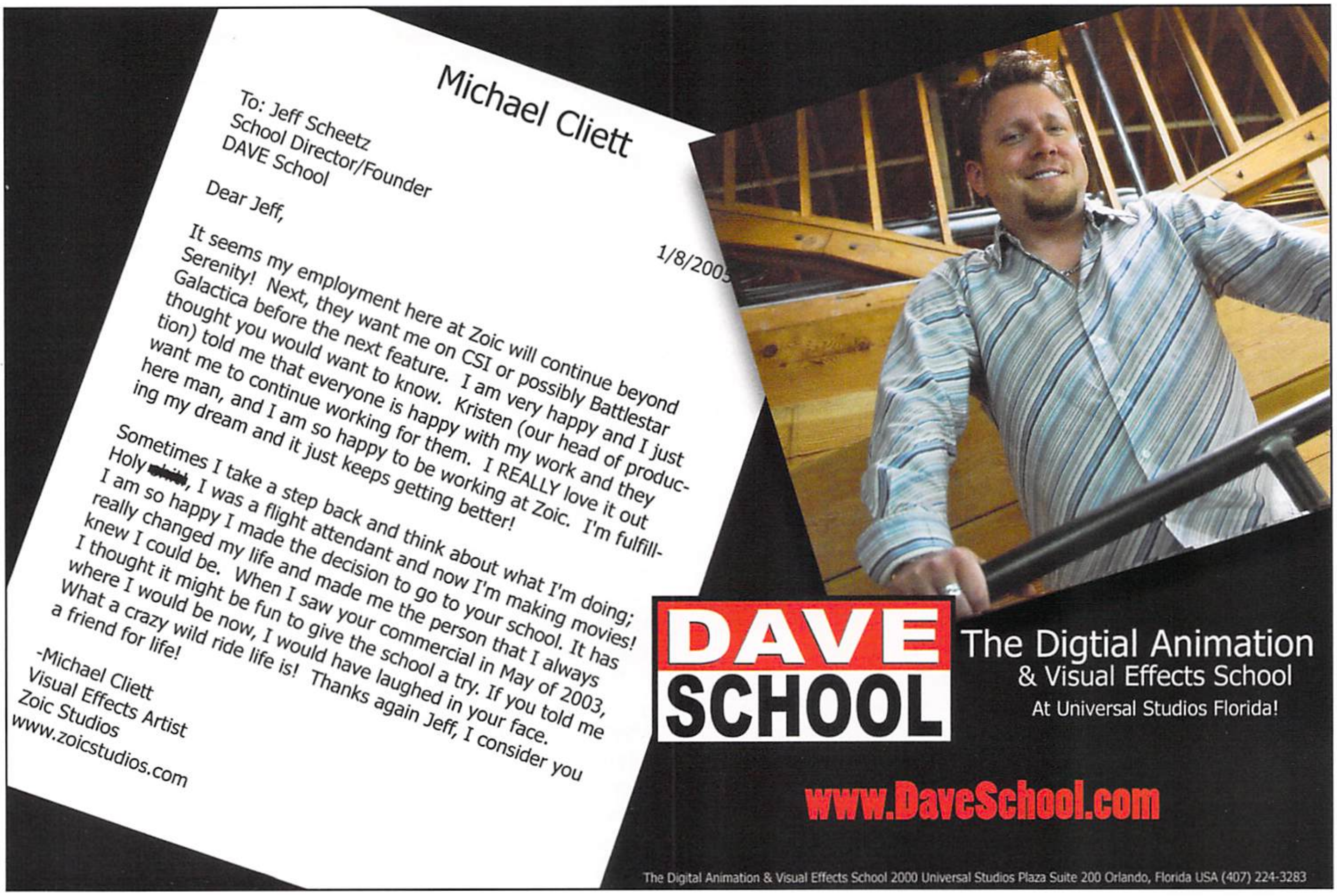
COMING NEXT TIME

That's it for now, kids. Next time, we'll look at the actual process of writing and debugging your scripts, and at making interfaces so your script is really cool and people actually won't mind using it. In the meantime, remember:

- **Use shortcuts only rarely**
- **Make your code human-readable**
- **Give your variables descriptive names**
- **Use comments liberally**
- **All commands end in a semicolon**
- **Always start counting with zero**
- **Declare your variables before you use them**
- **Use curly braces around code blocks even if they're only one line**
- **Make sure your loops won't go on forever before you run them**
- **Keep it simple, stupid.**
- **Save your work before you go messing with a script!**

Good luck, and always remember to have fun!

__Robin Scher__ has been animating for nearly ten years, and programming for over twenty, and is the author of the very popular Smedge render distribution system. He also enjoys taking photographs on occasion, and has just finished traveling around the world while programming the next version of Smedge. Right now, all he wants to do is to get some sleep.



Michael Cliett

To: Jeff Scheetz
School Director/Founder
DAVE School

1/8/2005

Dear Jeff,

It seems my employment here at Zoic will continue beyond Serenity! Next, they want me on CSI or possibly Battlestar Galactica before the next feature. I am very happy and I just thought you would want to know. Kristen (our head of production) told me that everyone is happy with my work and they want me to continue working for them. I REALLY love it out here man, and I am so happy to be working at Zoic. I'm fulfilling my dream and it just keeps getting better!

Sometimes I take a step back and think about what I'm doing; Holy [redacted], I was a flight attendant and now I'm making movies! I am so happy I made the decision to go to your school. It has really changed my life and made me the person that I always knew I could be. When I saw your commercial in May of 2003, I thought it might be fun to give the school a try. If you told me where I would be now, I would have laughed in your face. What a crazy wild ride life is! Thanks again Jeff, I consider you a friend for life!

-Michael Cliett
Visual Effects Artist
Zoic Studios
www.zoicstudios.com

DAVE SCHOOL
The Digtial Animation
& Visual Effects School
At Universal Studios Florida!

www.DaveSchool.com

The Digital Animation & Visual Effects School 2000 Universal Studios Plaza Suite 200 Orlando, Florida USA (407) 224-3283

Particles and Hypervoxels

THEY GO HAND IN HAND

By Deuce Bennett
Freelance CGI Artist
Lancaster, Texas

Welcome back everyone. This time around, I think I'm just going to shoot you a couple of simple tutorials dealing with what I encounter all the time: particles and HyperVoxels.

In my old practical effects work for movies, a common thing was to lay out 'bullet hits' – small explosive devices hidden from the camera to kick out dust or debris to simulate bullets striking the ground or environment from gunfire. Doing this in LightWave is so much more fun and much less work!

Then, we'll discuss a new feature to HyperVoxels in 8.3 dealing with those particles.

OK – so, let's get to it!

First – let's create an emitter that will be our "gun." Since we're not really interested in what kind of emitter (HyperVoxel or partigon), let's just stick with "HyperVoxel" because it will give us faster feedback while we're working (Fig 1).

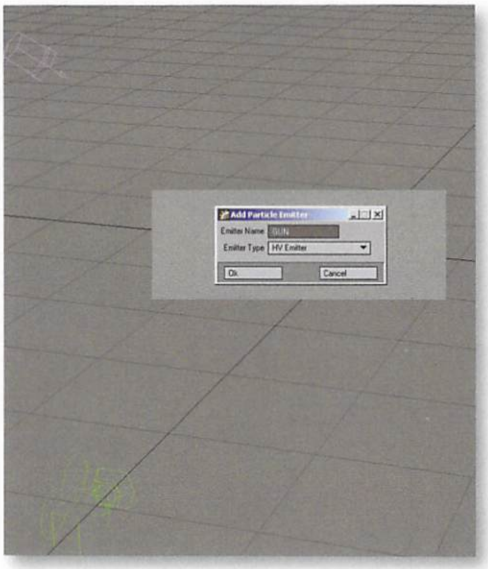
Now, our gun doesn't have to be too big, so let's just keep it simple to shoot down the Positive Z-Axis. We also don't really need a machine gun to explore this with, so keep the firing rate down (Figs 2 & 3).

We need something to shoot against. This is where we use a collision object (Fig. 4).

For this exercise, just a simple box-shape collision object is fine. In your own use, different shapes, or even the object that you have modeled, could be the collision object (Fig. 5).

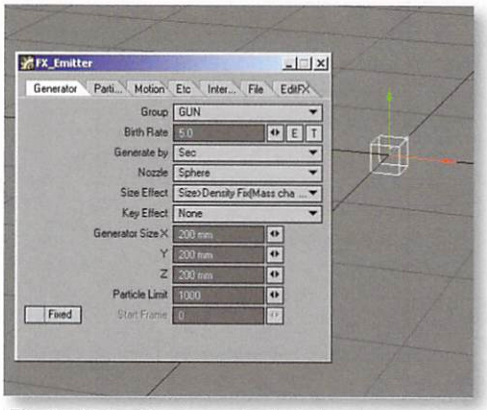
One thing here – I set up the "Gun" with the group name "Gun," and need to make a distinction here in the collision object to only erase "Gun" rounds. This is for my particular use; you might experiment with others.

Now, I typically use a setting of "erase" because I don't want my bullet object to hang around after hitting the wall. Get creative– I used bounce to make a stone skip across a lake. It's all in what you're after and the particular use that you need.



Add Particle Emitter
Emitter Name: GUN
Emitter Type: HV Emitter

Fig. 1– Granny get your gun.

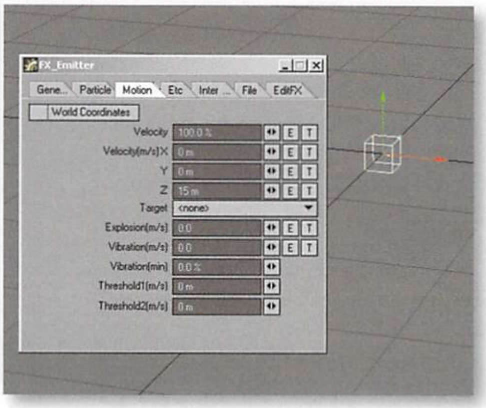


FX_Emitter

Group: GUN
Birth Rate: 5.0
Generate by: Sec
Nozzle: Sphere
Size Effect: Size>Density Fix(Mass cha ...
Key Effect: None
Generator Size X: 200 mm
Y: 200 mm
Z: 200 mm
Particle Limit: 1000
Fixed
Start Frame: 0

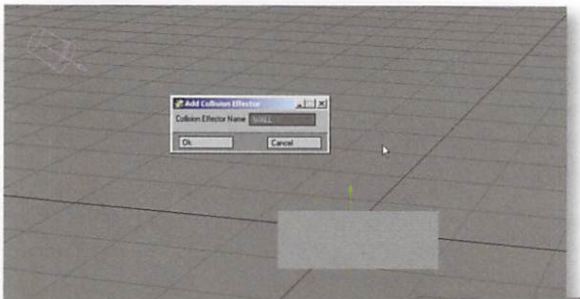
Fig. 2– Not a big gun, not a fast shooter.

Fig. 3– Shoot down the Z



FX_Emitter

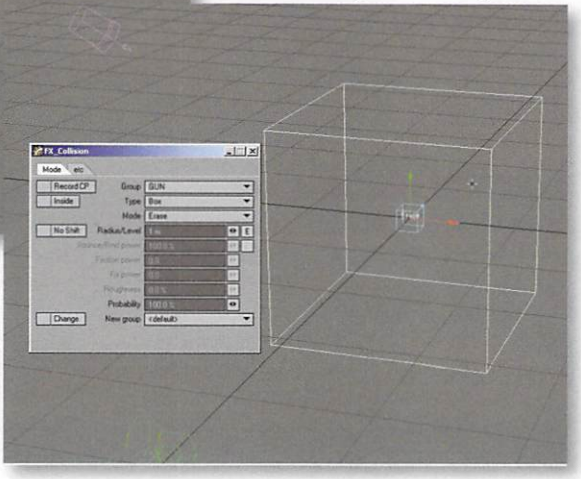
World Coordinates
Velocity: 100.0 %
Velocity(m/s)X: 0 m
Y: 0 m
Z: 15 m
Target: <none>
Explosion(m/s): 0.0
Vibration(m/s): 0.0
Vibration(min): 0.0 %
Threshold1(m/s): 0 m
Threshold2(m/s): 0 m



Add Collision Effector
Collision Effector Name: WALL

Fig. 4– Hello Wall.

Fig. 5– Erase my bullets.



FX_Collision

Record CP
Inside
Group: GUN
Type: Box
Mode: Erase
No Shift
Radius/Level: 1 m
Probability: 100.0 %
Change
New group: <default>

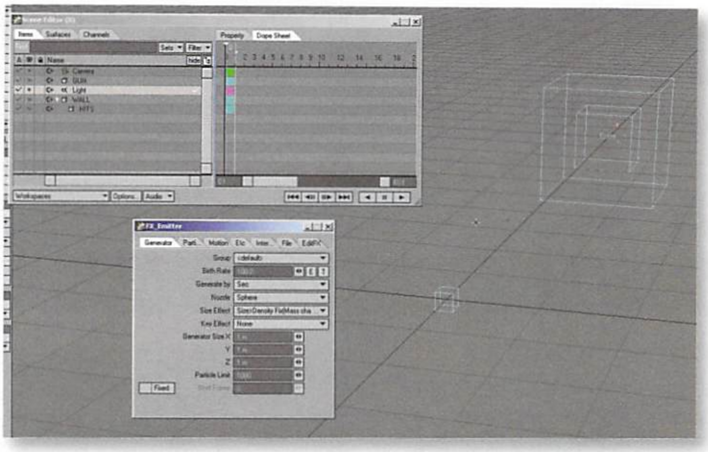
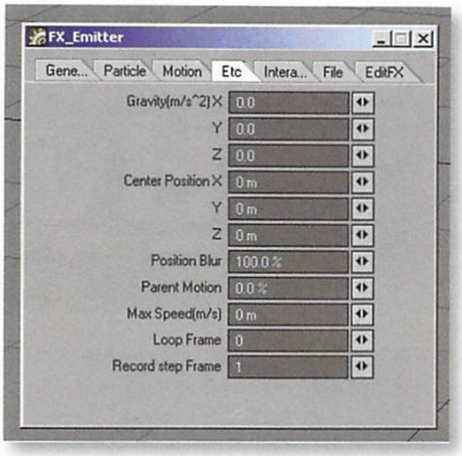


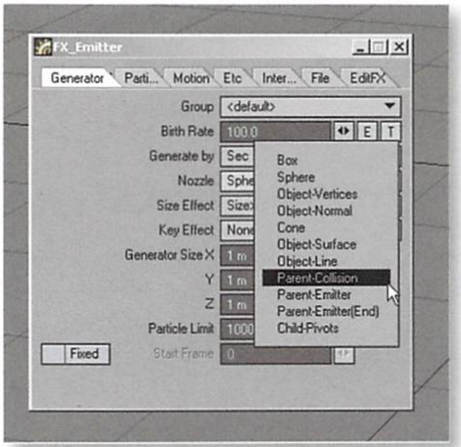
Fig. 6– Put the hits in the wall.



FX_Emitter

Gravity(m/s^2)X 0.0
Y 0.0
Z 0.0
Center Position X 0 m
Y 0 m
Z 0 m
Position Blur 100.0 %
Parent Motion 0.0 %
Max Speed(m/s) 0 m
Loop Frame 0
Record step Frame 1

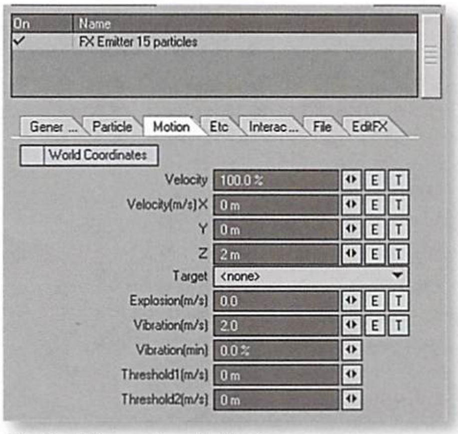
Fig. 7– Hits do their own thing.



FX_Emitter

Group <default>
Birth Rate 100.0
Generate by Sec
Nozzle Sphe
Size Effect Size
Key Effect None
Generator Size X 1 m
Y 1 m
Z 1 m
Particle Limit 1000
Start Frame 0

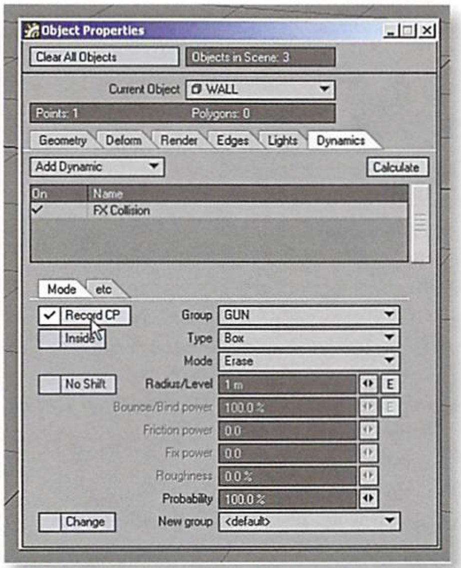
Fig. 8– Hits happen where the parent gets hit.



On Name
✓ FX Emitter 15 particles

☐ World Coordinates
Velocity 100.0 %
Velocity(m/s)X 0 m
Y 0 m
Z 2 m
Target <none>
Explosion(m/s) 0.0
Vibration(m/s) 2.0
Vibration(min) 0.0 %
Threshold1(m/s) 0 m
Threshold2(m/s) 0 m

Fig. 9– Hits go Z and vibrate.



Object Properties

Objects in Scene: 3
Current Object WALL
Points: 1 Polygons: 0

On Name
✓ FX Collision

☑ Record CP
☐ Inside
☐ No Shift
☐ Change
Group GUN
Type Box
Mode Erase
Radius/Level 1 m
Bounce/Bind power 100.0 %
Friction power 0.0
Fix power 0.0
Roughness 0.0 %
Probability 100.0 %
New group <default>

Fig. 10– Record where you're hit.

So, we have our gun shooting the wall. Now we add the 'hits'.

Create another emitter, and parent it to the collision object (Fig. 6).

Now, first and foremost, for this particular effect, we do not want to inherit the velocity of the bullets. So, we turn that attribute either off totally or way down. Otherwise, the "hits" will shoot through the other side of the wall, definitely NOT what we want (Fig. 7).

We want the emitter to be "Parent Collision," so we choose that from the Nozzle dropdown. This gives us an emitter for the 'hits' that is parented to the collision object. Thus "Parent Collision" – or 'where the parent gets hit.' (Fig. 8)

Since we turned the "Inherit Parent" velocity off or down, we need to tell it which direction we **do** want them to go. I did a -3 down the Z with a little vibration (Fig. 9).

One last thing we need to set or it just won't work: on the Collision Properties, we need to activate "Record CP" – this stands for "Record Collision Points," and means for the collision object to literally 'record' where it is being struck. This is being passed along to the emitter that is a child of our collision, to respond with the "Parent Emitter" setting (Fig. 10).

Well, that is really all there is to that; now you should be able to see how this worked in setup and come up with some interesting uses. Strafing airplanes, simple tracer rounds to hits, ground hits, wall hits – just about any of the bullet hits that had me crawling around with a shovel to physically simulate, you can now do just by animating an emitter. Consider yourself lucky (Fig. 11).

OK – that was an easy one, but useful. You know, if you put in some gravity, and made the gun bounce instead of erase, you could skip across a pond … hmmmm, might try that!

HyperVoxels, of all modes, have always lacked a certain connection with particles. This lack drove me crazy for many, many years. They purged the demon with the release of LightWave 8.3. Let's discuss this happy-happy joy-joy connection called "Relative Particle Age."

I have two emitters set up, with not-so-interesting settings, except one on each – the Random Particle Age setting. I have them both set to a 60 frame life span, plus or minus 30 frames. So my particles could live as long as 90 frames, or as short as 30 (Fig. 12).

Let's fire up HyperVoxels and make these sprites – you can use any kind, but for better OpenGL feedback, I use sprites, because no other HyperVoxels setting will display color information as clearly.

OK, on the emitter on the left, I made the particles 200mm, but only affected the color with a gradient using "Particle Age." Set each key to 25, 50, and 75 for the color changes (Fig. 13).

Now, look at the plume.

You'll see that only the particles that lived long enough (past 75 frames) received the black – and you can just make out almost "levels" where each color change happens.

I used the same settings for the emitter on the right, – but "Relative Particle Age" for the input (Fig. 14).

Looks completely different, doesn't it?

The color is reading the age of the particles independently. This means that the gradient input is in percentage of life – not hard and fast times. So every particle goes through the black, because that is at the end of its life. Also, you don't see the

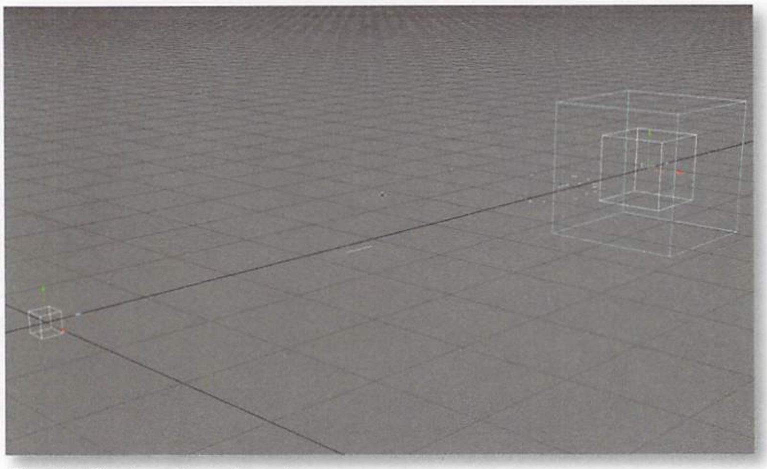


Fig. 11– Ouch ouch ouch ouch.

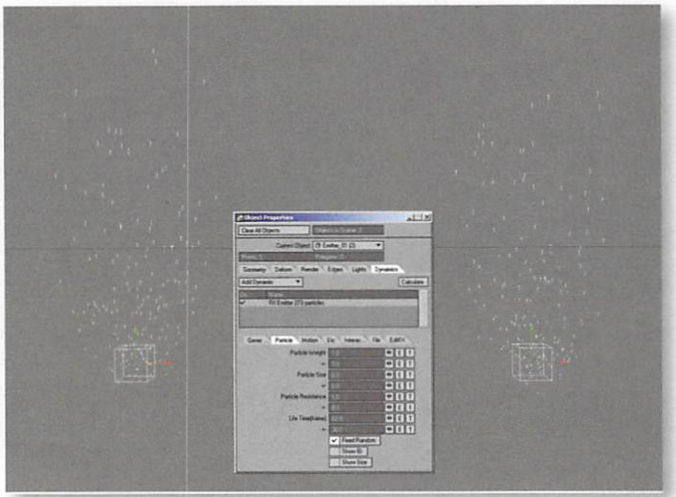
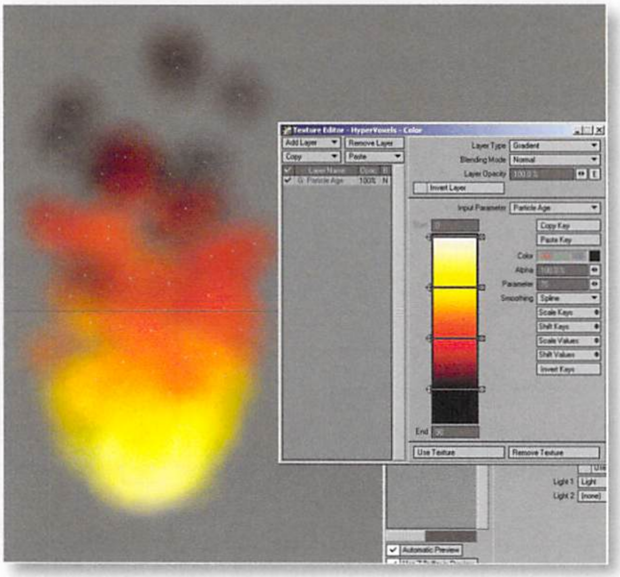


Fig. 12– Two Emitters



Texture Editor - HyperVoxels - Color

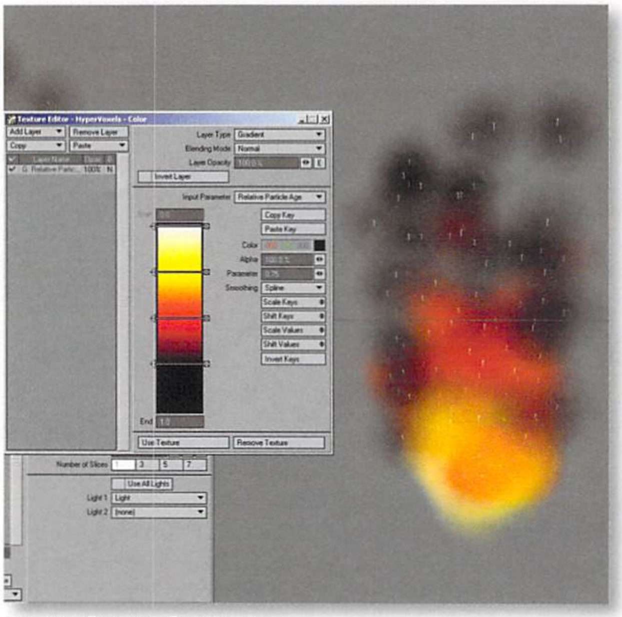
Layer Type: Gradient

Blending Mode: Normal

Input Parameter: Particle Age

Smoothing: Spline

Fig. 13– Particle Age



Texture Editor - HyperVoxels - Color

Layer Type: Gradient

Blending Mode: Normal

Input Parameter: Relative Particle Age

Smoothing: Spline

End: 1.0

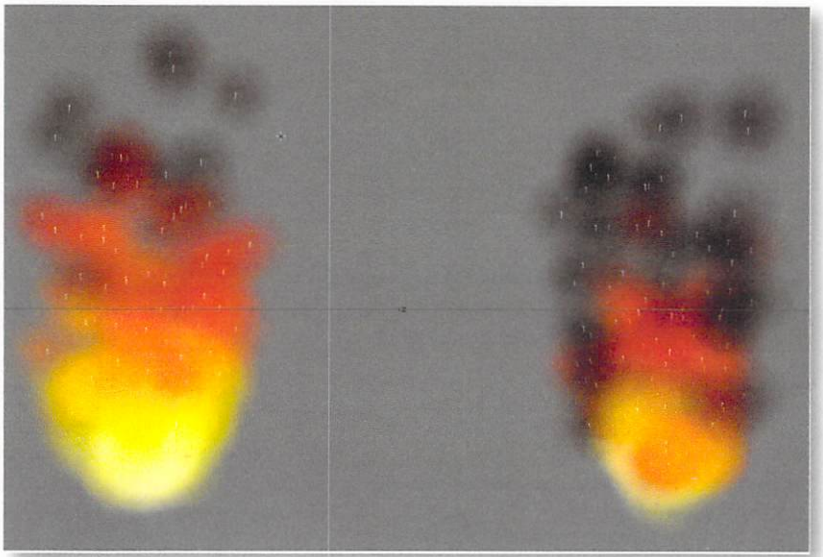
Fig. 14– Relative Particle Age

'banding' like in the other. This is much more random, and better for many cases. This gradient works for all settings. Use it for dissolve to avoid the "pop" at the end of a particle lifespan. Grow them over their life, shade, opacity changes – it's all there. This one simple update has made my life so much easier (Fig. 15).

Thanks for reading! Keep on waving – and maybe I'll see you at conventions. I do get out sometimes. ;-)

***Jack "Deuce" Bennett II** is a freelance CGI artist with a background in physical special effects for motion pictures and television. Deuce has been working in the film industry his entire life, and has such movies as **Robocop, Lonesome Dove**, and **Jimmy Neutron: Boy Genius** to his credit, as well as TV shows like **Walker, Texas Ranger**. Deuce has been using computers since he was 9, and started off writing his own graphic programs. He is a unique combination of physical knowledge and virtual know-how.*

Fig. 15– Ahhhhh

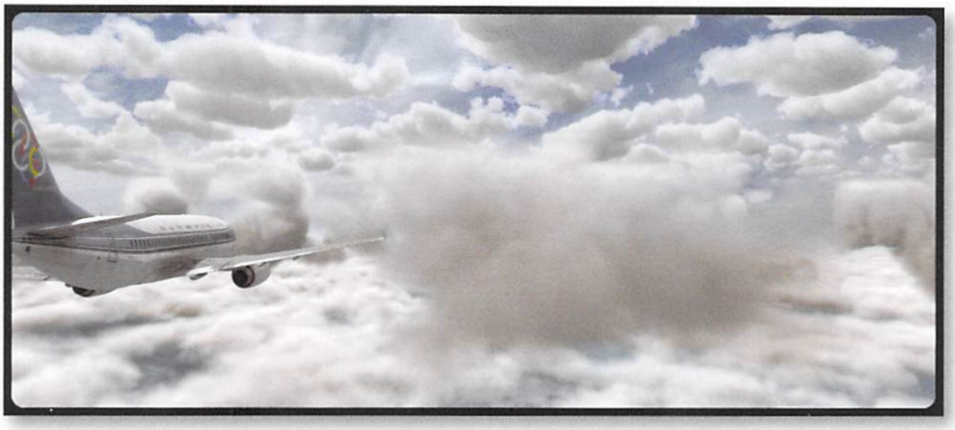


Color
BC
Lens Distortion
Merge Ground
Depth Gradient
Depth Blur
Horizon Glow
Color Corrector
Particles Displace
Merge Center
BC for Particles
Airbus320
Clouds Shadow
Back
Matte Control
Excausts
Luma Keyer

Compositing 101

Layers in Digital Fusion

By Gregory Glezakos
3D Artist, Teacher
Athens, Greece.



Pic02– Frame from the breakdown walkthrough.

Introduction

A rendered image from any 3D software needs to get some kind of tweak one way or another from editing or compositing software in order to be considered finished, unless, of course, there is a specific reason not to. These editing and compositing programs range from Adobe Photoshop to eyeon's Digital Fusion, Apple Shake and the top of the price range like Discreet Flame, Inferno, and the like.

In this tutorial, we'll see the flexibility we have when working with layers produced from any favorite 3D application, in order to deliver a better final version of our vision. Digital Fusion is the application we'll use, but the theory is exactly the same for all the compositing software available on the market.

Working with layers

Now, there are a lot of reasons to work in layers for a 3D scene that we can mix later using compositing software. Some of the most obvious are:

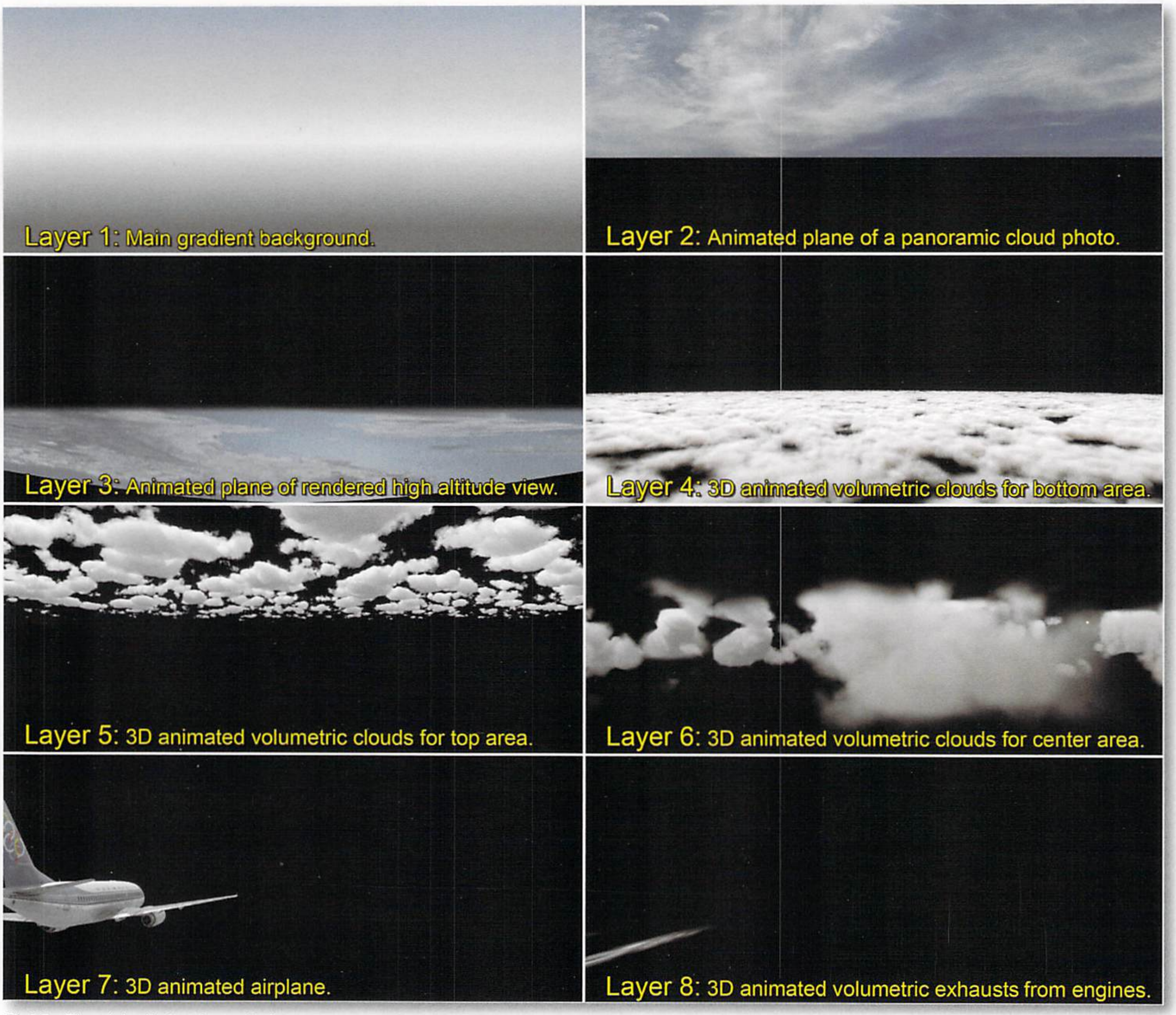
1. Easier manipulation of the 3D assets that assemble our 3D scene. It's easier to deal with one object at a time in our scene, especially if we have objects that don't interact with one another.
2. If our 3D scene is heavy with high polygon or subdivision surfacing objects and high resolution textures, chances are that our computer will come to a crawl, if not crashing all the time, trying to deal with all those assets at once. Again, our best bet for a scene like this is to break it up in layers.
3. Having our scene's objects in different layers, we can easily color correct or add motion blur on each one individually. We can do things in post in a fraction of the time it would take us to make such changes through a re-render of our image, or even trying to do all post work in a single rendered frame.

So, unless we're dealing with a single object in a single color background, we should consider creating our final image through layer compositing (Pic02).

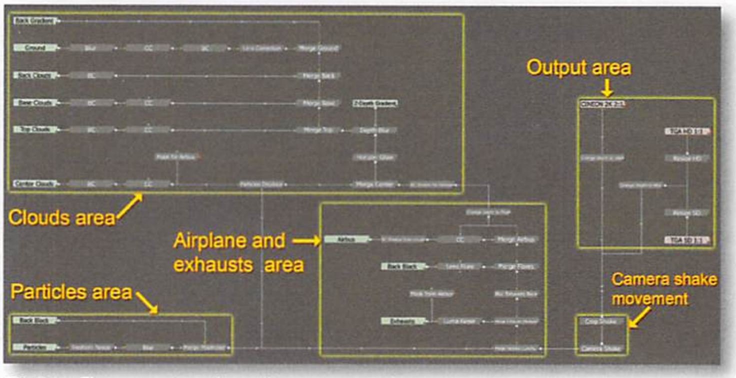
Let's break up the scene from the above image and see why it's easier to work with layers. This scene was a shot for a feature film; I rendered it in 2K anamorphic resolution, 1828 x 1556 pixels in 2:1 aspect ratio. Quite a beast, considering that it required the camera to fly inside volumetric 3D clouds and was 850 frames long. I decided early on in the production to break up the scene into distinct layers for compositing back later to produce the final image sequence. I used Maya's Fluids for the center cloud layer and LightWave for everything else. I created the 3D volumetric clouds for the top and bottom layer using the OGO-Taiki plug-in. Digital Fusion was the compositing software, and in Pic03, we see the distinct layers as they look before compositing.

Go with the Flow

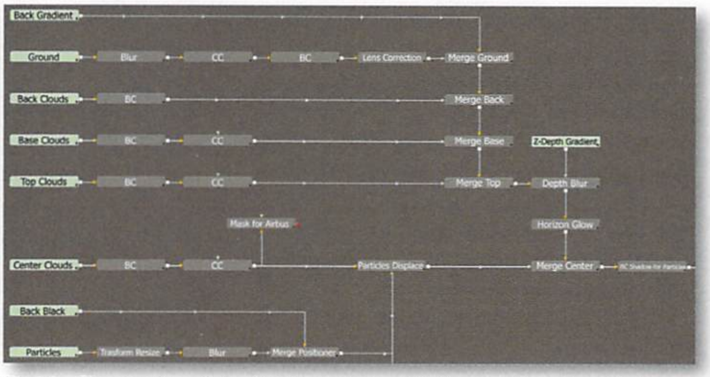
In Digital Fusion, we build our flow starting from the lowest layer in the scene – the background – to the outmost layer – the aircraft and its exhausts. In Pic04, we see the complete flow at a glance. Each layer is loaded in Digital Fusion and renamed accordingly. Renaming loaders, tools, masks, and everything in Digital Fusion is a good habit to keep things as organized as possible. In Pic05, we see the portion of the flow that deals with the background and all the cloud layers; I've made the names of each tool bigger in Photoshop for easier viewing. Looking more carefully at each line of the flow, each layer is altered from its original rendered form at the loader, to its final settings at each


Layer 1: Main gradient background.
Layer 2: Animated plane of a panoramic cloud photo.
Layer 3: Animated plane of rendered high altitude view.
Layer 4: 3D animated volumetric clouds for bottom area.
Layer 5: 3D animated volumetric clouds for top area.
Layer 6: 3D animated volumetric clouds for center area.
Layer 7: 3D animated airplane.
Layer 8: 3D animated volumetric exhausts from engines.

Pic03– The eight distinct layers before any compositing takes place.


Output area
Clouds area
Airplane and exhausts area
Particles area
Camera shake movement

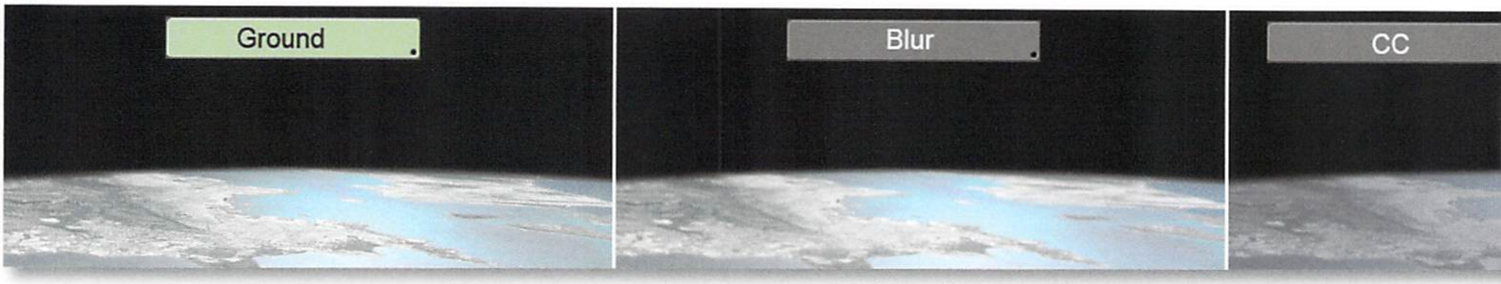
Pic04– The complete flow diagram at a glance.


Back Gradient
Ground
Blur
CC
BC
Lens Correction
Merge Ground
Back Clouds
BC
Merge Back
Base Clouds
BC
CC
Merge Base
Z-Depth Gradient
Top Clouds
BC
CC
Merge Top
Depth Blur
Mask for Airbus
Horizon Glow
Center Clouds
BC
CC
Particles Displace
Merge Center
Back Black
Particles
Transform Resize
Blur
Merge Positioner

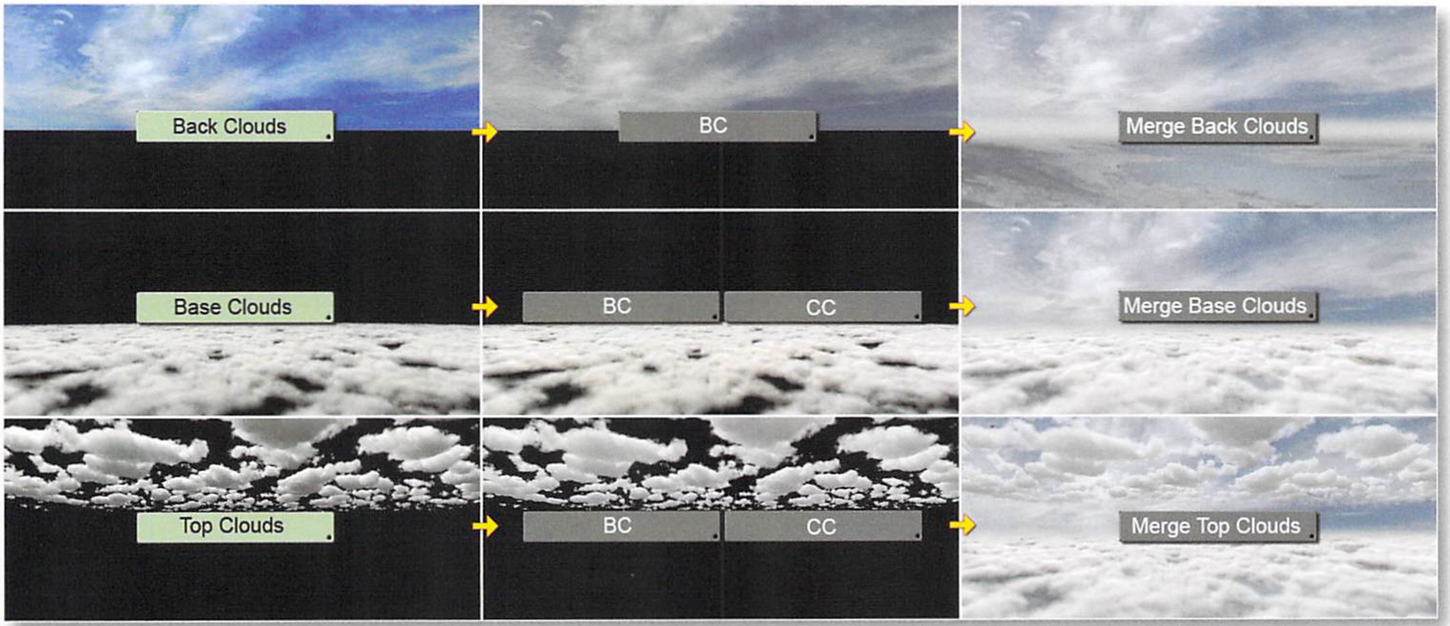
Pic05– The portion of the flow that deals with the clouds and particles.

Merge tool, through the use of tools like Brightness/Contrast (named BC on the flow for easier viewing), Color Correctors (CC), Blurs, and the like.

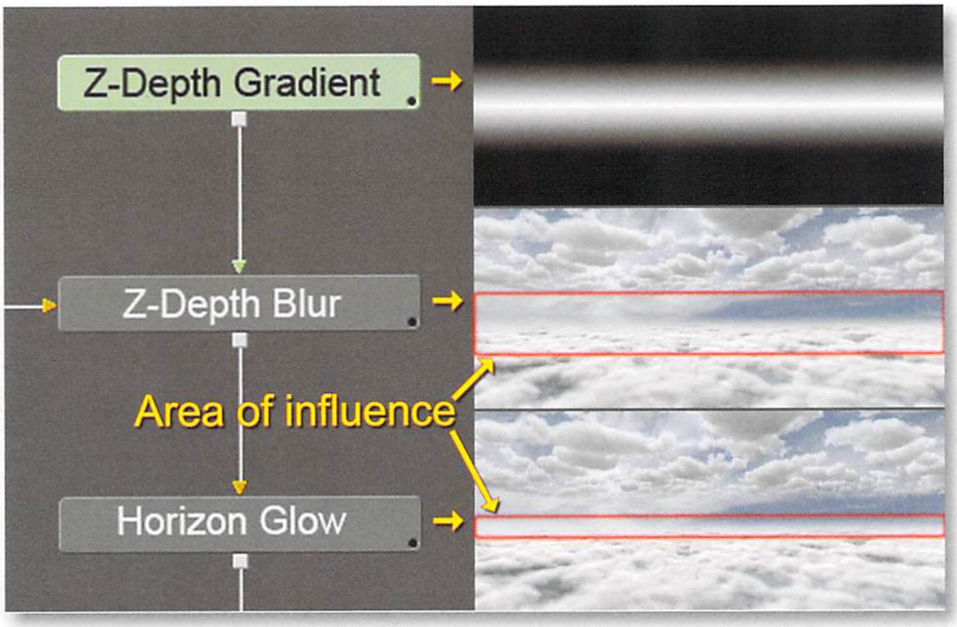
The first layer, which is the lowest layer of all, is a Background tool with a simple gradient, as seen in Pic03. It won't be seen after all the layers are placed above, so it doesn't need any special treatment. Its main purpose is to define the flow's resolution, which for this case is 2K anamorphic or 1828 x 1556 pixels in 2:1 aspect ratio. Second layer is the ground layer, a slowly moving 2D plane with a texture map of a photo taken from high



Pic06– The Ground layer and how it evolves through the flow.



Pic07– The Back, Base, and Top cloud layers through each tool of the flow.

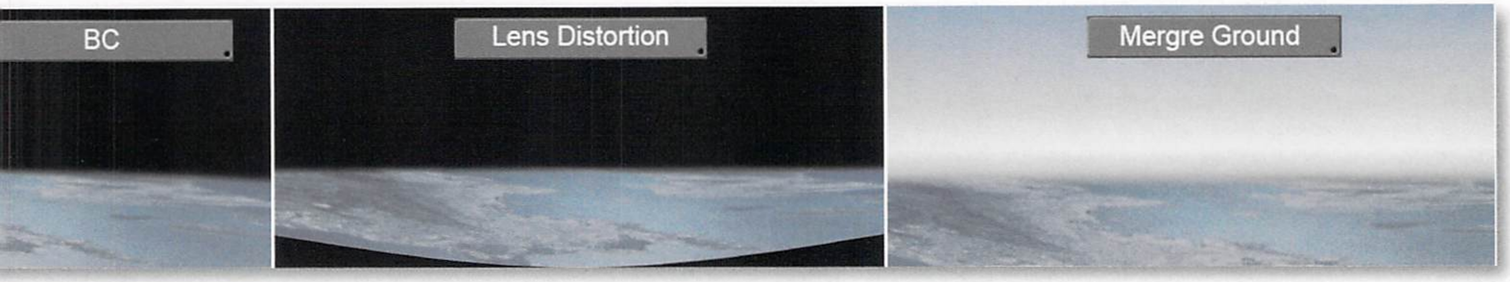


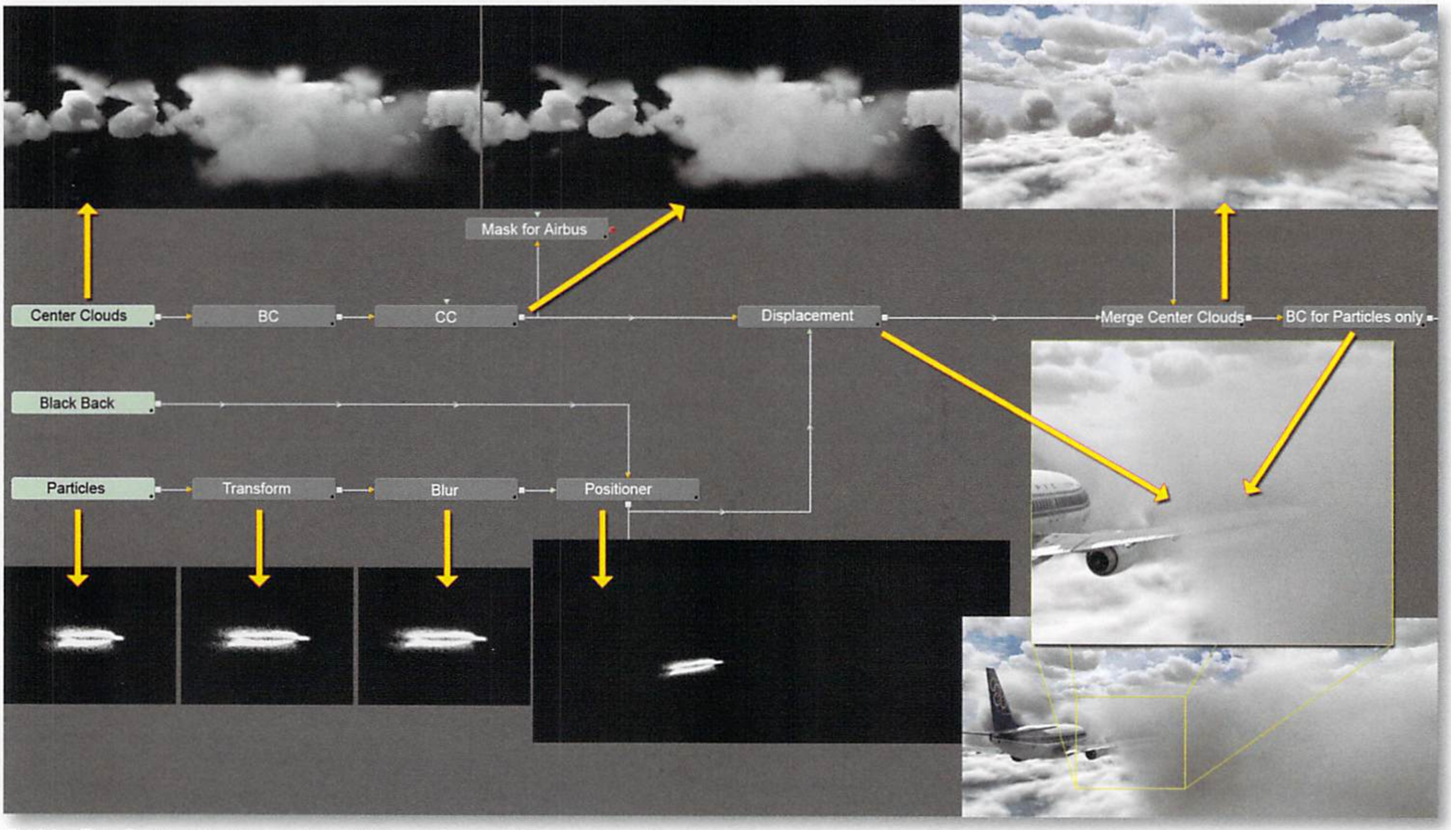
Pic08– Faking Z-Depth defocusing effect and glow from the distant horizon.

altitude, rendered in LightWave. For the ground layer, I used an extra tool to remove the unwanted fish eye lens distortion that occurred in render time. The ground layer won't be seen much, mostly only portions of it, so we won't spend a lot of time with it. In Pic06, you see how it evolves from the default rendered image at the "Ground" green loader to its final state at the "Merge Ground" tool, where we've also scaled the layer up a bit to remove the lower area in black. In a similar manner, Pic07 shows the treatment of the Back, Base, and Top cloud layers in the flow. Basically, throughout the flow, we use the Brightness/Contrast tool to make sure that each layer has the correct light intensity, and with the Color Corrector tool, we balance the colors to make sure the scene has the mood we want to achieve. At each Merge tool, we add a rectangle mask to softly blend each layer at the horizon line.

As you see, working in post allows us do great things easily, unlike in a 3D software program. For example, for the horizon line, we can add a fake Z-depth blur by adding a black & white gradient and use that as a luminance area of influence for the Depth Blur tool in Digital Fusion. Also, we can add a soft glow only at the horizon line, using a rectangle glow mask affecting that area only (Pic08).

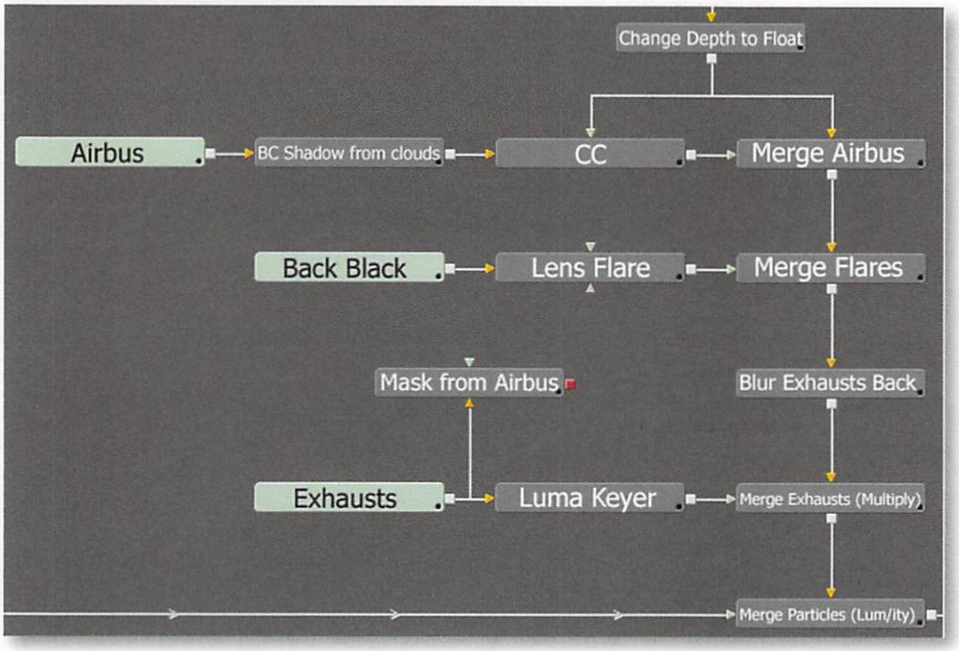
Moving down the flow, we arrive at the center cloud layer. We do the same as before with this layer, with one extra treatment. This is an animated sequence we're working on, so towards the end of it, the airplane's right wing is passing through a cloud from this layer. We've got to introduce a gentle cut in the cloud. We do this by using an animated sequence of a particle layer, which we've made in LightWave, but which it is possible to make using Digital Fusion's built-in


BC
Lens Distortion
Mergre Ground


Mask for Airbus
Center Clouds
BC
CC
Displacement
Merge Center Clouds
BC for Particles only
Black Back
Particles
Transform
Blur
Positioner

Pic09– The Center cloud layer, the collision effect particles, and the flow process.

Pic10– The Airplane and Exhausts layers flow diagram.

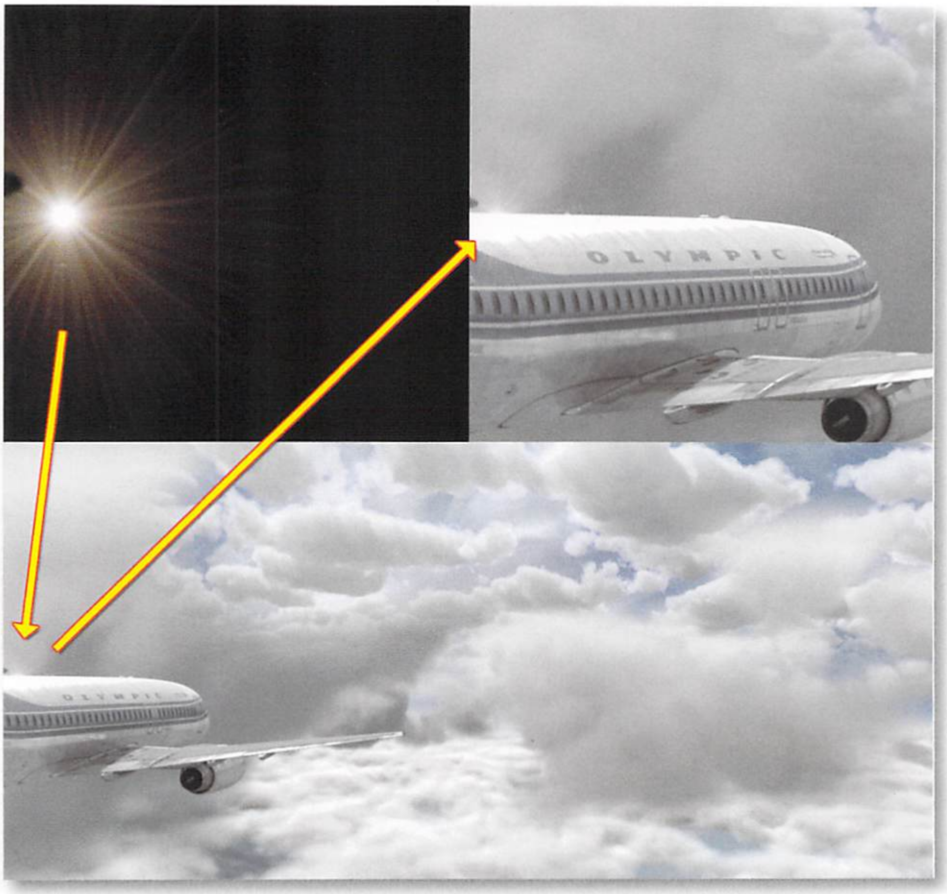

Change Depth to Float
Airbus
BC Shadow from clouds
CC
Merge Airbus
Back Black
Lens Flare
Merge Flares
Mask from Airbus
Blur Exhausts Back
Exhausts
Luma Keyer
Merge Exhausts (Multiply)
Merge Particles (Lum/ity)

particle system. We create this particle sequence, simulating the vortex behavior of a particle cloud when an object similar to an aircraft's wing cuts through it. Then, in Digital Fusion, we resize and blur this layer a bit and use a Merge tool to position it where the right wing seems to pass through the cloud (Pic09). The trick is to use a Displacement tool to deform the center cloud layer at that position and use a Brightness/Contrast tool to create fake shadows on top of that cloud layer, by using the particle layer as a luminance mask; Pic09 shows the process. Also, since we place the aircraft layer on top of all the cloud layers, we use a Matte Control tool (Mask for Airbus) to create an animated polygon mask for moving the aircraft's wing behind the cloud. This happens towards the end of the sequence, so we animate the ins and outs accordingly to make this happen at the specific frame we want.

Next, we deal with the aircraft. In Pic10 (previous page), we see the portion of the flow that describes ITS compositing process. Besides the usual Color Corrector tool, we've added a Brightness/Contrast tool, which we use with a polygon mask to create fake shadows from the clouds onto the airplane in order to create the illusion that it flies among them. Also, we added a Lens Flare tool to enhance the highlight of the sun onto the top side of the airplane's fuselage the moment it enters the frame (Pic11). This flare fades out shortly after, as the airplane travels forward. Then, we add the Exhausts layer, using a multiply blend for the Merge tool; a small percent value is used. Now, before that tool, we add a Blur tool, which we use with a luminance mask from the Exhausts layer, to blur everything underneath in order to simulate the heat displaced from the hot exhausts. On top of all that, we add the Particles layer I mentioned before, using a luminosity blend through a Merge tool.

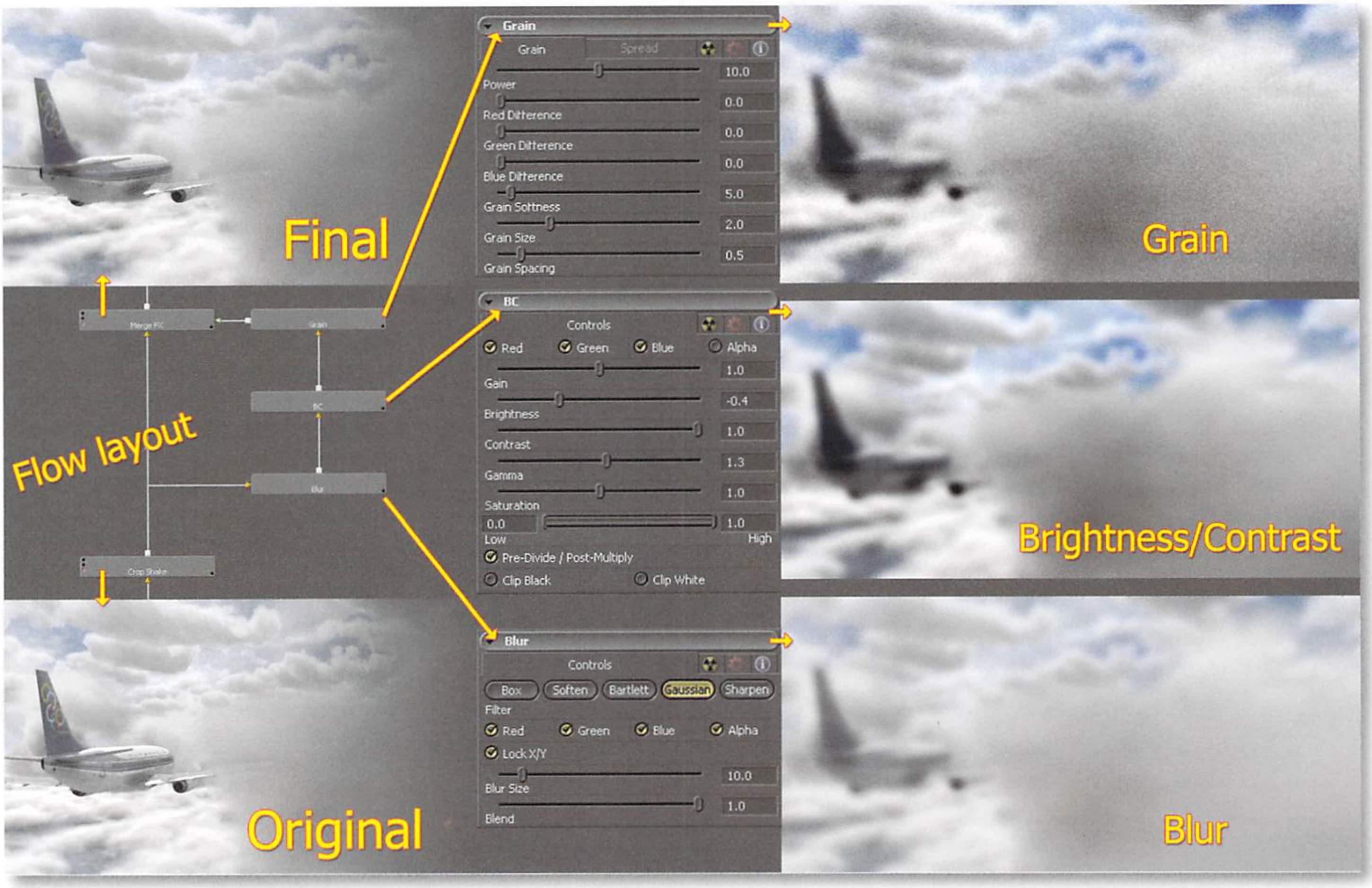
Finishing Touches

The layer compositing is done, so now we add some finishing touches to the



Pic11– Adding a gentle flare boost to the specular highlight for the airplane.

Pic12– The blurring, contrast, grain, merge effect in action.



Grain

Grain | Spread

Power 10.0

Red Difference 0.0

Green Difference 0.0

Blue Difference 0.0

Grain Softness 5.0

Grain Size 2.0

Grain Spacing 0.5

BC

Controls

☑ Red ☑ Green ☑ Blue ○ Alpha

Gain 1.0

Brightness -0.4

Contrast 1.0

Gamma 1.3

Saturation 1.0

Low 0.0 High 1.0

☑ Pre-Divide / Post-Multiply

○ Clip Black ○ Clip White

Blur

Controls

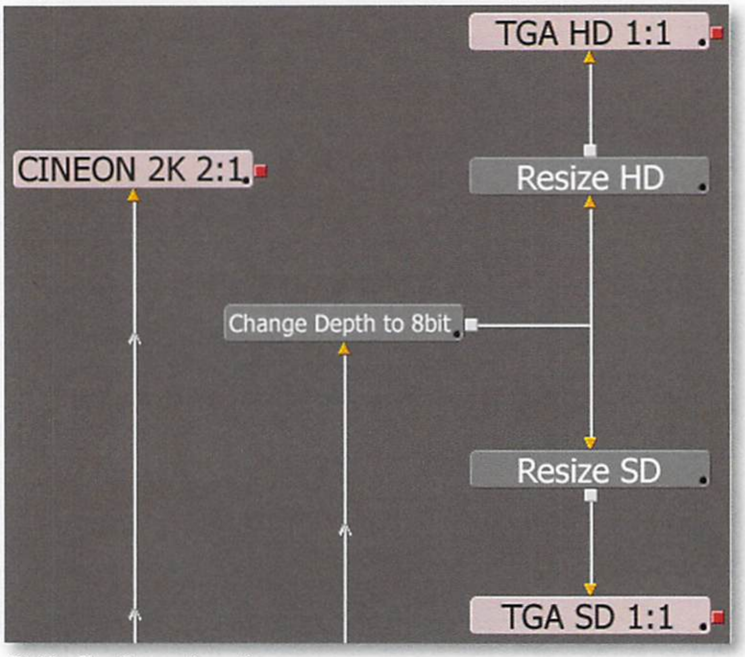
Filter: Box Soften Bartlett [✓ Gaussian] Sharpen

☑ Red ☑ Green ☑ Blue ☑ Alpha

☑ Lock X/Y

Blur Size 10.0

Blend 1.0



Pic13– The Savers flow diagram.

The final composited image

overall sequence to complete the project. We've added a Camera Shake tool after the airplane flow area to enable us to add slight motion variation to the already animated sequence; this creates the illusion that we are actually inside the other aircraft that is filming the hero aircraft, which is not flying straight and level. Then, we have to add a Crop tool to zoom into the image, eliminate any blank areas at the corner's frame that the Camera Shake tool will introduce, and rescale the image up to the correct proportions.

Now, since this project was going for print in film, we didn't want to add film grain, as the printing process will introduce film grain either way. Normally, if your target media is not film or it is for film and you have to blend CG imagery within live action, then you have to add some kind of film grain to get that "film look" or to properly match the live action sequence.

Here's a great trick to immediately remove that 3D look from our images is the following steps:

1. Add a Blur tool with input from our final image (so far) and use the Gaussian option with a value of 10.
2. Next, add a Brightness/Contrast tool with settings for Brightness -0.4, Contrast 1 and Gamma 1.3. We want to add a lot of contrast to our image, but not alter the overall luminosity.
3. Then, add a Grain tool with settings you choose, depending on the final look you want to achieve. Common settings are 10 for Power, 5 for Grain Softness, 2 for Grain Size, and 0.5 for Grain Spacing.
4. Finally, use a Merge tool with a Normal blend value of no more than 0.25, to blend the image with itself.

In Pic12, we see the flow and the process, along with the settings for each tool. Experiment with each tool to suit your needs. Credit for the idea about this tip goes to the Toronto LightWave User Group site at: *www.lwhub.com/rendertrick/index.html* and a small section from the lighting course paper #30 from **SIGGRAPH '96.**

Last thing to do, of course, is set up the Saver tool to save the render of our flow. Since this particular project was going to film, we chose the 10-bit Cineon format. Alternatively, we used two more savers, both with a Resize tool, to convert the 2K 2:1 original resolution to High Definition (HD) and Standard Definition (SD) format; we selected an 8-bit Targa format for those two (Pic13).

Final Thoughts

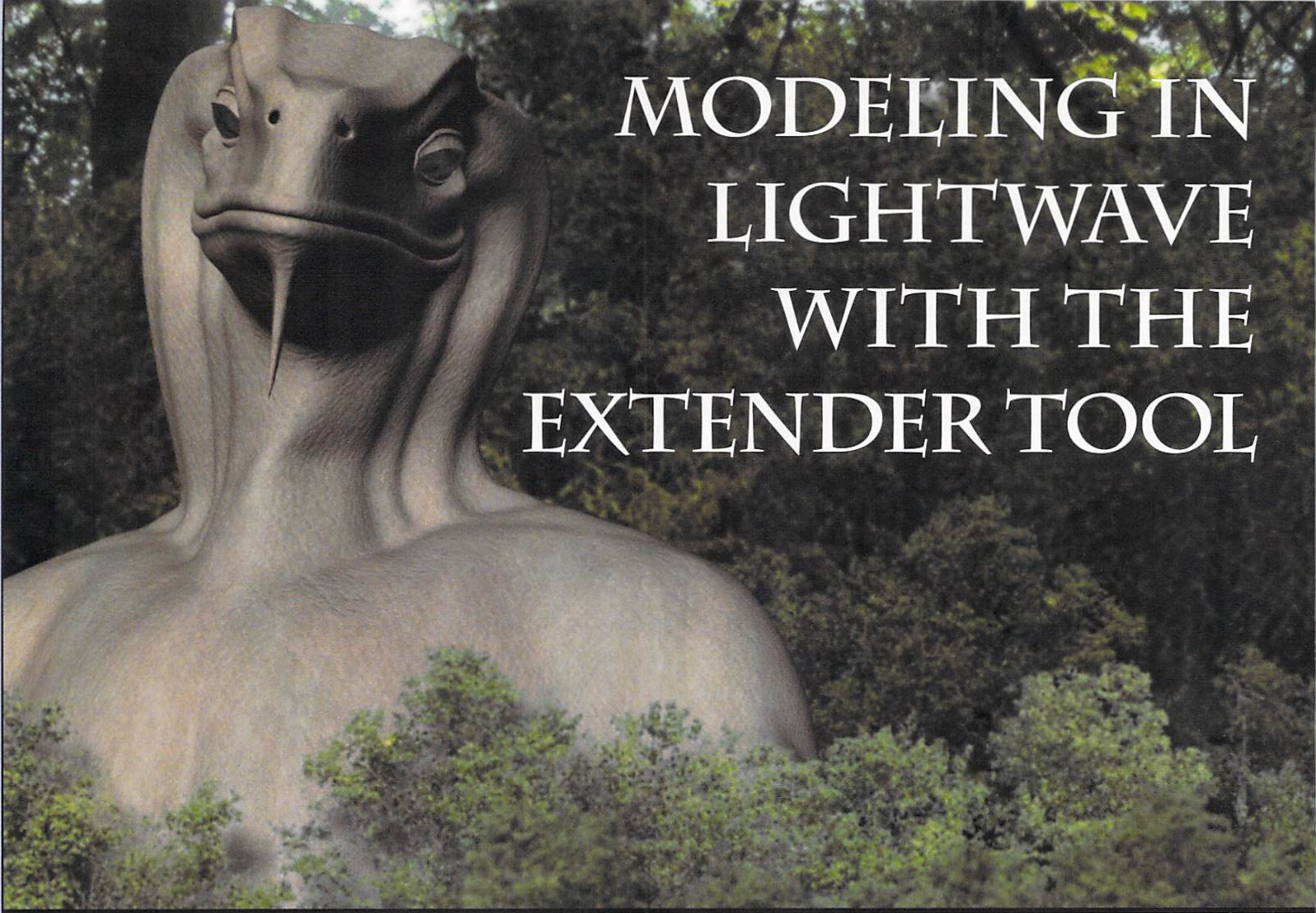
As you see, building our 3D sequence from distinct and different layers, instead of rendering everything at once, is by far much easier, more flexible, and gives us tremendous power and control over the look and feel of our creation. The same process is also effective when rendering our 3D objects in passes like Color, Diffuse, Specular, Reflection, etc. With passes, it's only a matter of the type of blend we use for our Merge tools, and their position in the flow, in order to rebuild the final result; again, with much more control than a single pass render.

I used Digital Fusion 4 for this breakdown walkthrough. Version 5, available soon, will change the way we do our everyday compositing, since it will be fully OpenGL oriented with true 3D environment built right into it. As I said before, the theory behind the process works with all available compositing software, and if it's for a still image, Photoshop or similar applications do the job, as well. I hope you find this walkthrough useful and that it helps you take your 3D and compositing work a step up ...

Gregory Glezakos *(aka "T.Rex" to his friends) is an experienced 3D artist working professionally for the last 10 years, either as a freelancer or contracted to major domestic production studios. When between jobs, he is teaching LightWave and Digital Fusion, either to students or by helping artists in various visual FX houses meet their deadlines.*

Tell us what you want in future issues.

Go to:
http://www.hdri3d.com/vote/index.htm



MODELING IN LIGHTWAVE WITH THE EXTENDER TOOL

By Bob Calandriello
Medical 3D Modeler
Animator
Totowa, New Jersey

Working for a medical device company is a good way to learn organic modeling techniques. This lesson is just one of many ways to model a complex figure using the Extender tool in conjunction with the Drag and Move tool. In this lesson, I'd like to create something a bit different from the medical field, but still model a human-type figure with a few variations, as you can see from the images. Since we work in the virtual world on our computers day in and day out, we can create anything that our imaginations can dream up. We'll concentrate mostly on the modeling aspect and touch slightly on texturing towards the end of the article. We'll start off by creating the eye, and work outward. It will be a high-polygonal model, for the simple reason of demonstrating how we can create the realistic muscle tone with polygons.

So let's start off by making the eye by adding some points and creating polygons from them as shown in **Figure 1**, and name them "Eye."

Next, select the outside edge points, and start using the Extender tool to move the points and polygons to shape the top and bottom eyelids by adjusting the points until you have something that looks like **Figure 2**, and name these polygons "Body."

We aren't going to make a separate eyeball for this model, since we'll color the eye a deep solid black. After you've

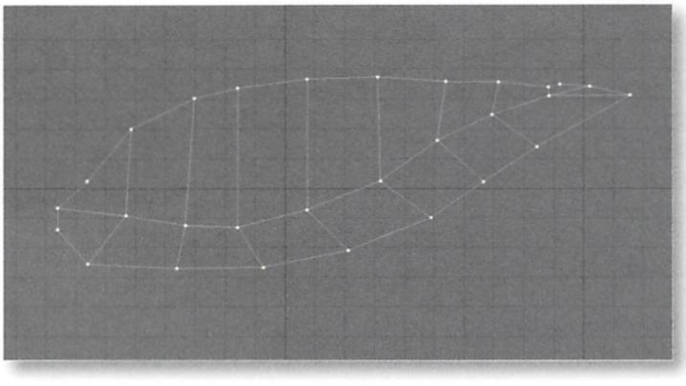


Figure 1

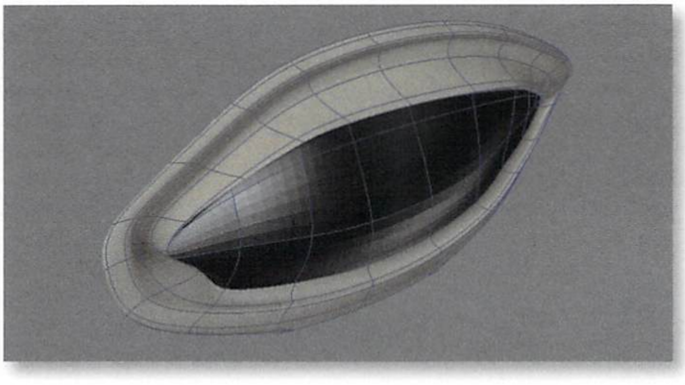


Figure 2

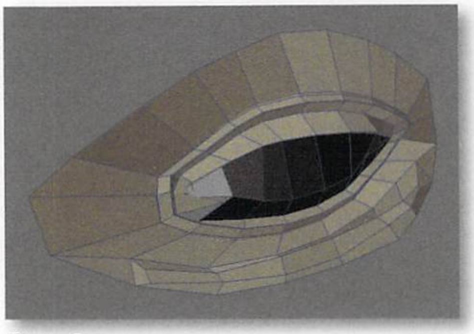
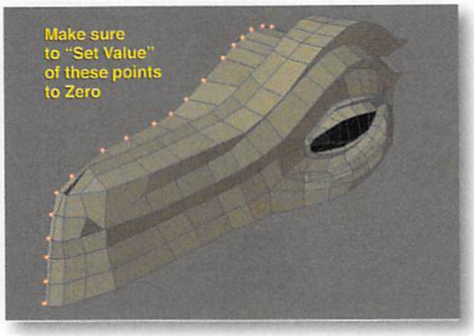


Figure 3



Make sure to "Set Value" of these points to Zero

Figure 4

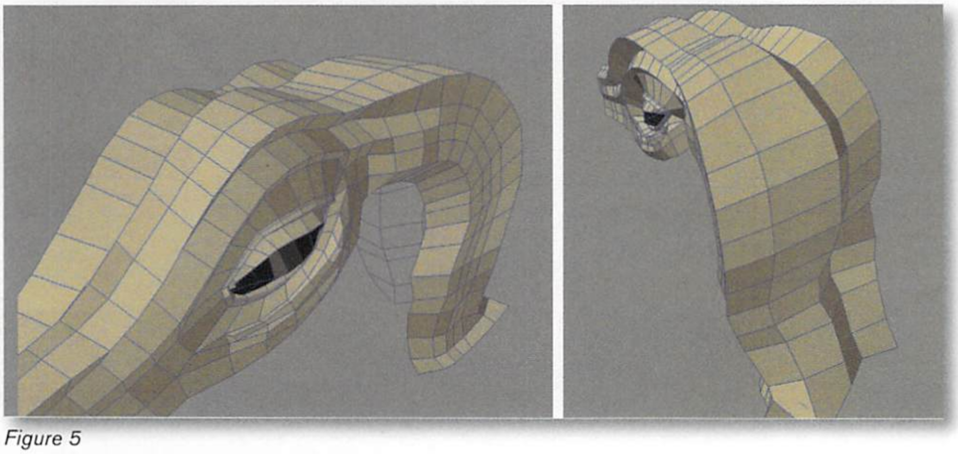


Figure 5

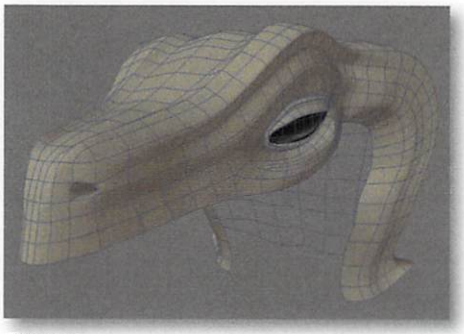


Figure 6

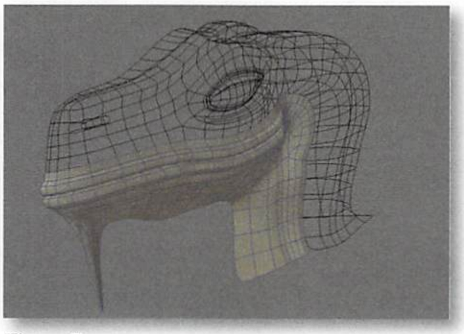


Figure 7

shaped the eye, create the eye socket surrounding area as shown in **Figure 3**.

Continue with the Extender tool to create the shape of the front and top of the head as shown in **Figure 4**. We are going to model half of this model, and then when the entire half is complete, we will Mirror the geometry. Since we are only modeling half, we must set the value for the last set of points to 0. The "Set Value" window is under the *Detail Tab > Points > Set Value*. Most of the time, before I even get close to finishing, I mirror the geometry just to see the progression, then undo the mirror and continue on if satisfied with the model.

Since we'll be creating a snake-type head, let's continue by building the back of the neck area. Once again, select the outside points, use the Extender tool, and shape the points and polygons to resemble **Figure 5**. Note the widened back of the head is slightly cobra-shaped. Before we continue on, mirror the geometry, then subdivide it to check if you like where your model is going **(see Figure 6)**. If you are happy with it, undo the subdivision and mirror command and continue on.

Now let's move onto the jaw region. We'll make the jaw by creating the polygons as shown in **Figure 7**. Also thought I'd give him a goatee.

Next, create the back as shown in **Figure 8**. As you can see, I once again mirrored the geometry to check the progression of the model. I like this mirror checkpoint. It's much easier to check it in stages, rather than go all the way back and make changes once you've completed the entire geometry you will be mirroring.

After the back area is finished, create the front of the chest area **(Figure 9)**, again using the Extender and Drag tool.

Okay, now to give this Lizard-Human Hybrid a set of Six-Pack Abs! Since our model's a reptile and a human, his

Figure 8

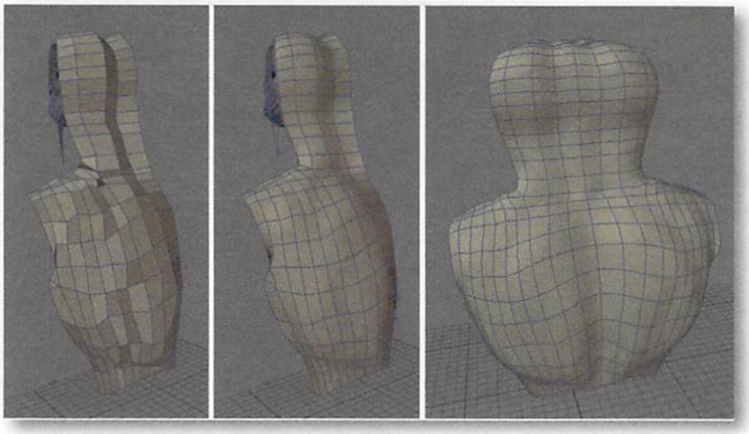
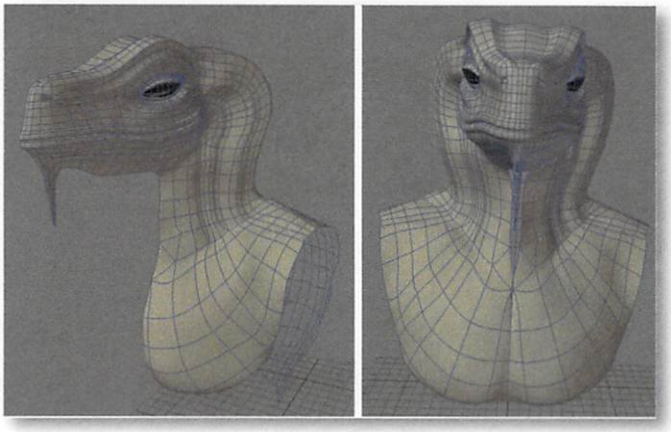


Figure 9



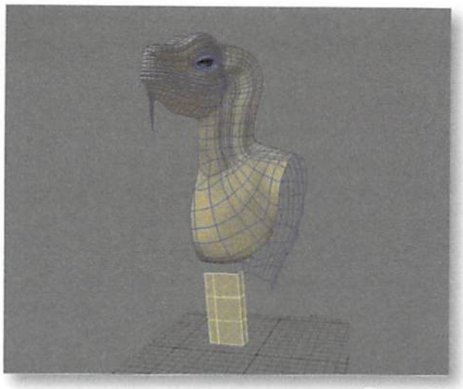


Figure 10

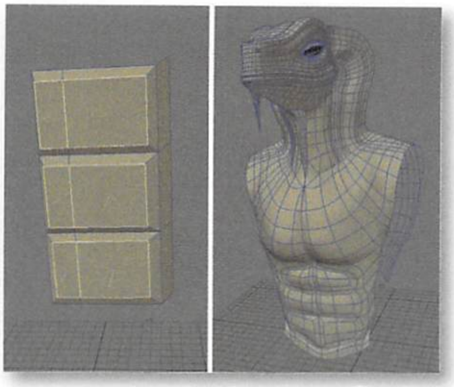


Figure 11

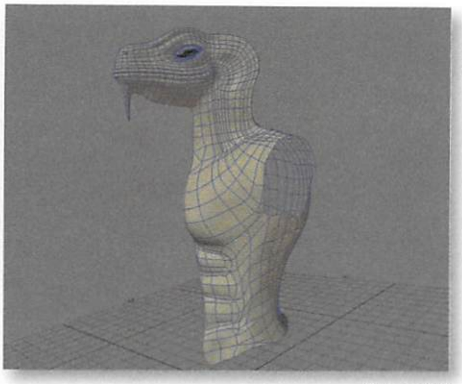


Figure 12

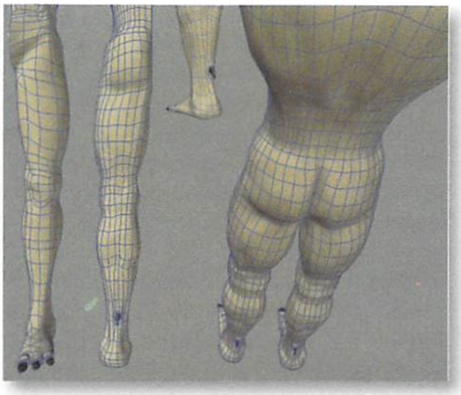


Figure 13

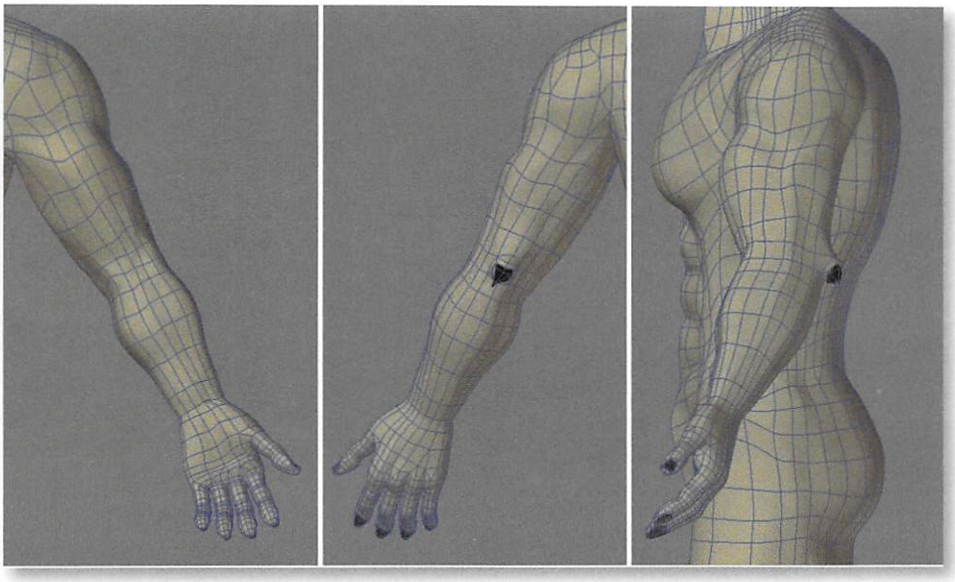


Figure 14

abs will reflect a bit of both. Create a box with three segments and "Smooth Shift" the segments as shown in **Figure 10**. From this point, start selecting the points, creating polygons that join to the rest of the figure, then go back to the Extender tool to create more of the stomach area, as shown in **Figure 11**. Note the Mirror check again. Create the side of the stomach now, connecting it to the back area as in **Figure 12**.

Continue to create the bottom half of the model **(Figure 13)** the same way we created the head and upper half of the body. When creating the knee and toes, use the "Smooth Shift" command. Try adding some spurs coming out of the back of the leg.

Create the arm and hand to complete the model **(Figure 14)** before we mirror the geometry. When creating the arm and hand, use the same Extender tool technique. Use the "Smooth Shift" command to model the fingernails.

Now that we've completed half the figure, mirror the geometry for the full figure **(Figure 15)**, and check on the creation of all your polygons, as in **Figure 16**. An easy fix: select three or four points and create a polygon.

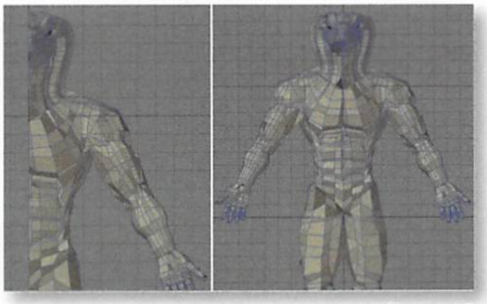


Figure 15

There are many ways of creating a human or animal figure, such as using spline modeling, polygonal modeling (which I prefer), or using the Extender tool. This was a long exercise, but once finished, you'll be very familiar with this versatile tool.

I surfaced this quickly by going onto the Internet and grabbing a few images of lizard skin and leather for texturing **(Figure 17, 18 and 19)**. As you can see in **Figure 20**, I used these files for texturing the model. See **Figure 21** for the values of each texture.

To surface this another way, I purchased a great plug-in called Eye Candy by Alien Skin Software (www.alienskin.com). This software enables you to create some pretty cool seamless textures, and is well worth the $99.00. I originally purchased it for the marble texture, which creates great bump maps for arteries and veins.

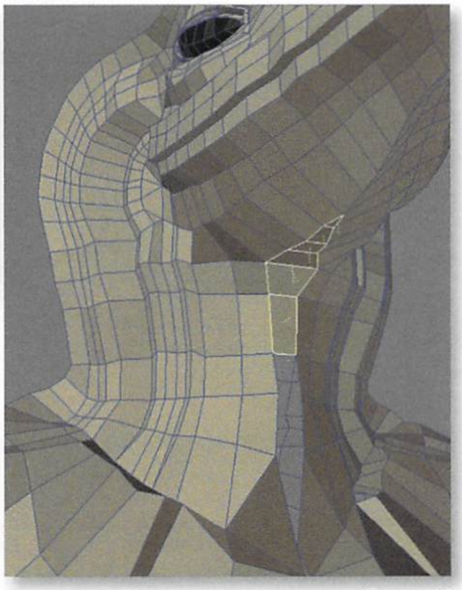


Figure 16

I used Eye Candy textures for **Figure 22** by creating a simple texture for the bump map, along with some Procedurals for the texture – it was that easy.

If you'd like to see your model come to life after all your hard work, check out

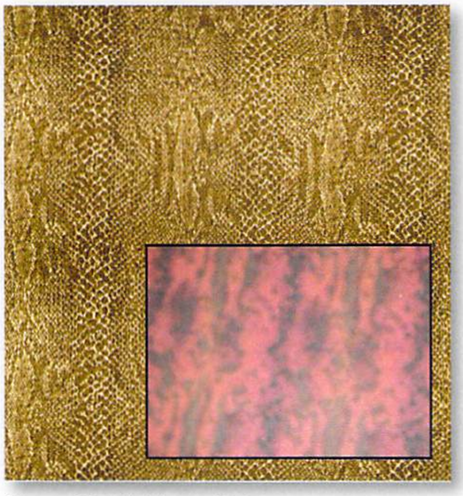
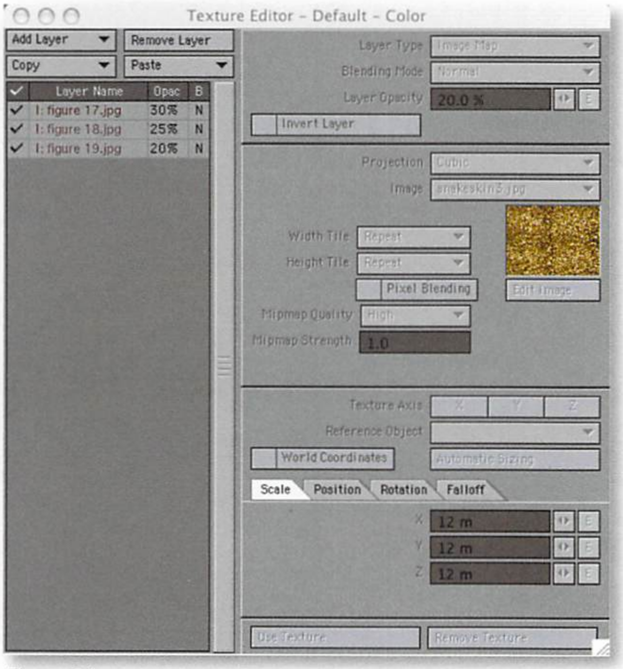


Figure 17 with Figure 18 Inset

Figure 19



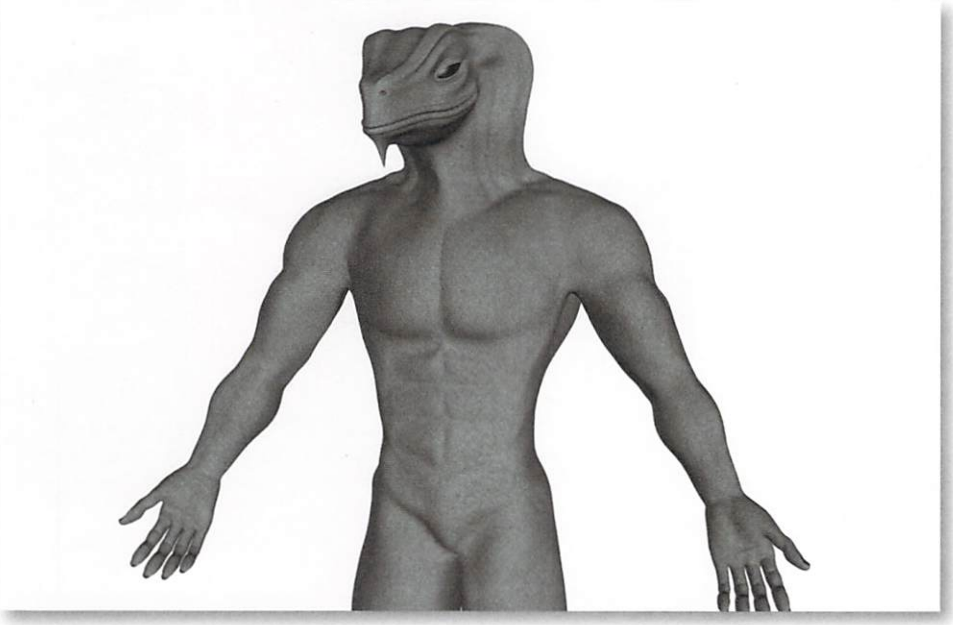
Texture Editor - Default - Color
Add Layer
Remove Layer
Copy
Paste
Invert Layer
Pixel Blending
Edit Image
World Coordinates
Automatic Sizing
Scale
Position
Rotation
Falloff
Use Texture
Remove Texture

✓	Layer Name	Opac	B
✓	I: figure 17.jpg	30%	N
✓	I: figure 18.jpg	25%	N
✓	I: figure 19.jpg	20%	N

Layer Type: Image Map
Blending Mode: Normal
Layer Opacity: 20.0 %
Projection: Cubic
Image: snakeskin3.jpg
Width Tile: Repeat
Height Tile: Repeat
Mipmap Quality: High
Mipmap Strength: 1.0
Texture Axis: X Y Z
Reference Object:
X: 12 m
Y: 12 m
Z: 12 m

Figure 21

Figure 22



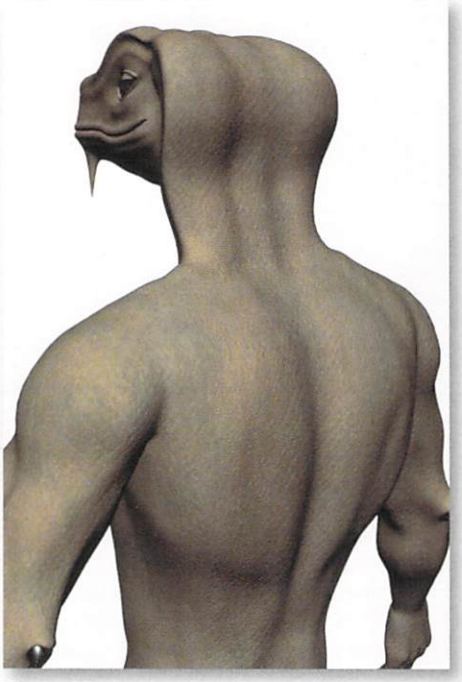


Figure 20

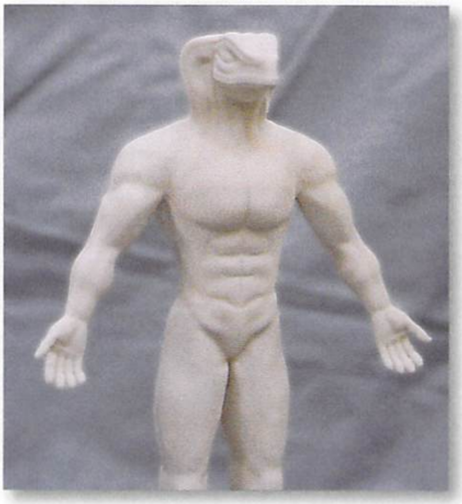
www.3dArtToPart.com. They accept a variety of different file types. I sent them the file I've written about in this article, and the price was very reasonable, and the staff was very helpful. This model cost $89.00, which included shipping! Not bad. It's so cool seeing a true "hard copy" of your virtual creation **(See model below)**.

Whatever you create, make it enjoyable. Once it becomes a job, move on.

Happy modeling!

Bob Calandriello *has spent the last several years working in the medical device industry. As graphics communications manager at cordis endovascular, a johnson & Johnson company, he works closely with sales and marketing, along with R&D, creating photo-realistic renders and animations for newly created medical devices. View more of his work at www.robertcal.com.*

The really cool hard copy model from our file.



SETTING UP a Face for Lip Synch in SOFTIMAGE|XSI

By Matt Morris
Animator, Rigger
Co-Founder of London XSI Users Group
London, England

This tutorial focuses on creating blend shapes and controllers within SOFTIMAGE|XSI, but it's worth taking a moment to examine some of the underlying principles and reasons for the approach used here.

Ever since the days of Disney, tackling lip synch animation means using phonemes, mouth shapes representing the various different sounds that form the alphabet. Preston Blair suggested around nine basic phonemes to produce correct lip synch: A/I, E, O, U, C/D/G/K, F/V, L, M/B/P, W/Q. The problem with using this method in CGI is that quite often it leads to overactive animation (trying to hit every syllable), or a very pose-based feel, as it hits the same shapes repeatedly.

Jason Osipa suggests a fundamentally different and more organic method of lip synch in his breakthrough 2003 book *Stop Staring*. He bases lip synch on 'visimes' rather than phonemes. Visimes, as the word suggests, concentrate on the visible shapes the mouth forms, rather than the audible sounds. The simplest setup using this method would include at the most basic: wide, narrow, closed, and open positions, and on top of this, you should also add top lip up/down and bottom lip up/down. Working this way stresses the importance of a shape relative to the neighboring shapes to help it read as desired. This method has the advantage of simplicity, though it sometimes lacks the finer detail a greater number of shapes would give you, so it works best as a base on which to build further shapes for the extremes/variations. In this way, you can create an effective combination of both methods, building extra shapes only when the animation demands it, and this will get you up and lip synching in the fastest time possible.

Before we start, make sure you have Helge Mathee's Image2Weightmap script installed on your machine, found either at *www.softimage.com* or *www.mindthink.de.*

To begin, create a symmetrical head, as seen in **Image 1**. The topology of the model is very important at this stage and you should be looking to create edgeloops that run around the contours of the mouth in order to form the wide range of shapes you will need. Place this under a model null; use of the mixer is extensive. I prefer to create a mesh with the mouth already closed; this can make enveloping a bit trickier, but it means you already have one shape ready (closed), and you can use this as a jumping off point for the other shapes.

The next step is to create a good UV map that is also symmetrical, or if you don't plan to texture it, a straightforward frontal XY projection will do the job. A tip to symmetrize both the head and UV map is to work up one side of the head until it's finished and UV unwrap that side until you are happy with the results. Making sure you delete the other incomplete side of the head, use poly select and hit F7 to draw around the remaining side and right-click on the head. From the dropdown list, choose 'Symmetrize Polygons', which creates the other side of the head with mirrored UVs. Selecting the polys on the new side of the head and entering the Texture Editor, scale the symmetrized UVs using the right mouse button until they reverse, as in the example in Image 2. You can then join the selection back to the original side by selecting the points down one side of the head and choosing the option 'Island Heal to Picked' and selecting the other side. Now, create a symmetry map to use when modeling the shapes – remember to make sure the head is lying at 0 on the global axis in X when you do this.

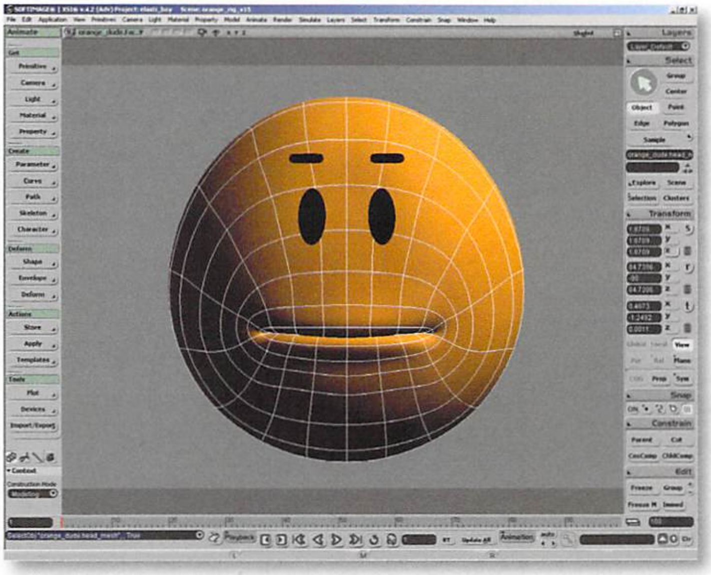


Image 1– Mesh with edgeloops running around the contours of the mouth.

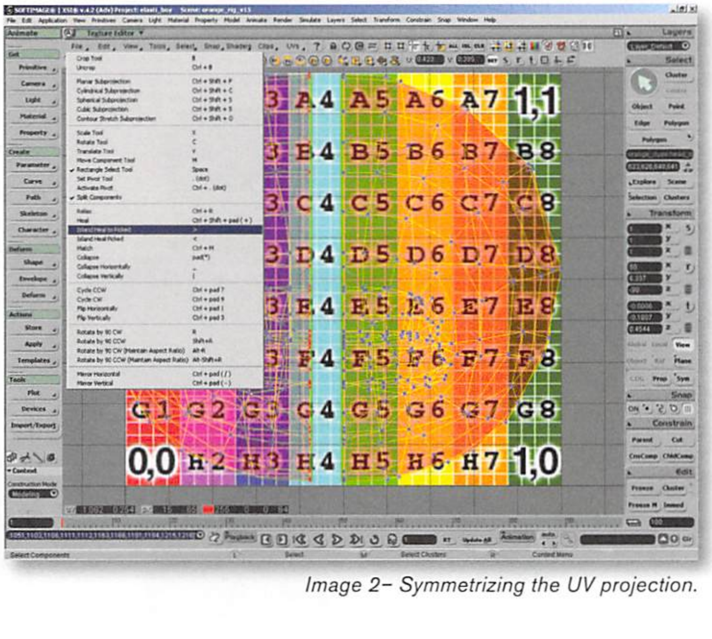


Image 2– Symmetrizing the UV projection.

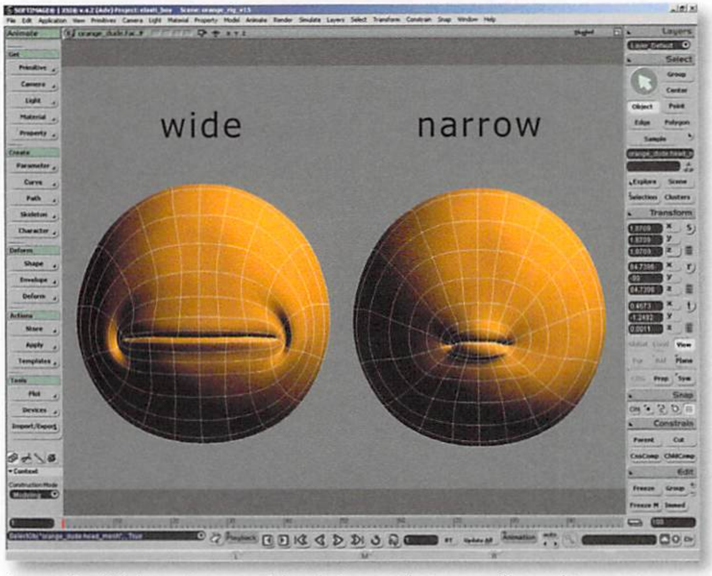


Image 3– Wide and narrow blend shapes that you will need to create.

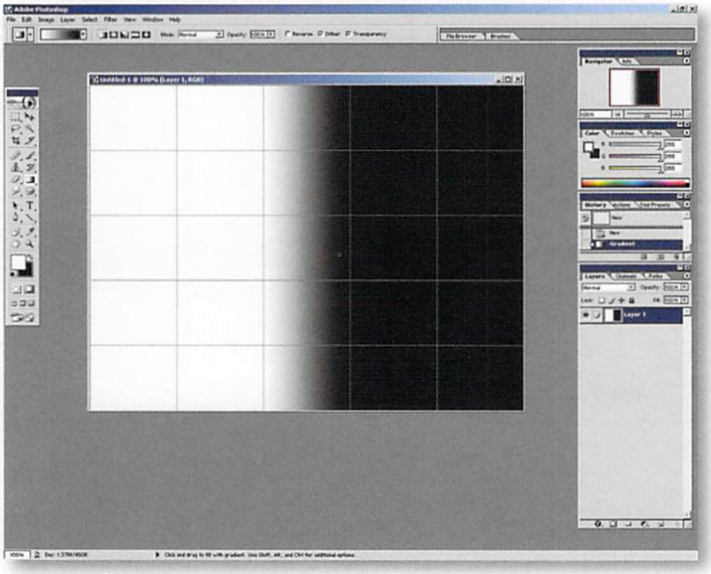
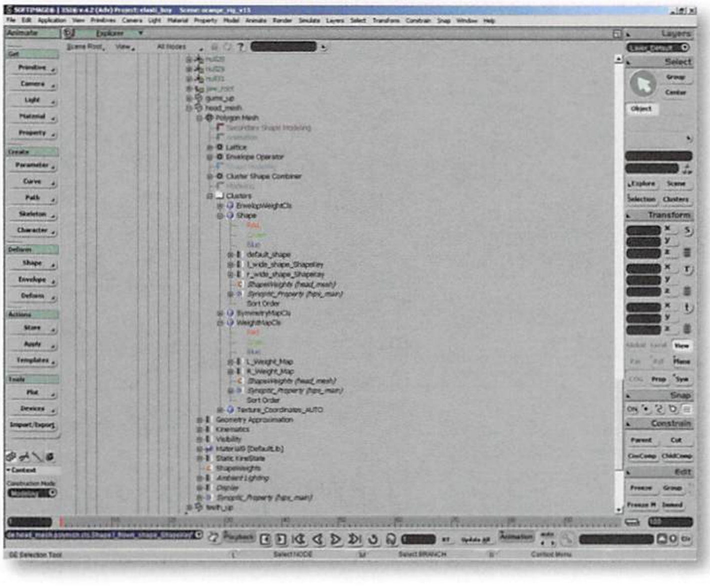


Image 4– Photoshop gradient.

Image 5– The shapes appear underneath the shape cluster of the main mesh.



Duplicate the head and create the shapes you need. For this example, I'm only creating wide and narrow shapes, shown in **Image 3**, as the open/closed positions will come from the jaw. The symmetry map created on the base mesh copies over also, so you can use that to make the shapes symmetrical. When you're happy with the shapes, freeze them to clear the op stack.

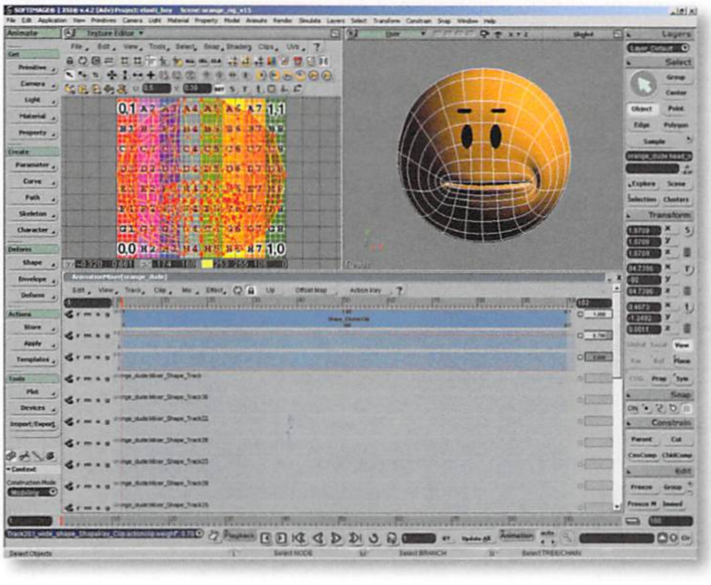
In Photoshop, create a horizontal black and white gradient that blends just through the middle of the image – see example in **Image 4**. It doesn't have to be very large, 600x400 will do, but it's worth experimenting if you have a highly detailed mesh. To make sure you have the gradient running directly through the middle of the image, turn on Snap to Grid, and in the Edit/Preferences/grid settings, place the grid lines at 20 percent and draw with the Gradient tool between the middle two vertical grid lines. Call this l_weight_map and save it as a file format that SOFTIMAGE|XSI won't have a problem with, such as targa. Flip it horizontally and save again as r_weight_map. Make sure you load these images into your scene clips memory by dragging and dropping them onto SOFTIMAGE|XSI.

Back in SOFTIMAGE|XSI, create a weightmap for your head, then select the mesh and run the ImagetoWeightmap script and select the l_weight_map image from the dropdown box. Freeze off the Image2weightmap module in the weightmap operator stack, and re-name the weightmap l_weight_map. Repeat the procedure for the r_weight_map, making sure you change the dropdown box for the weightmaps to the new one before applying the script again.

Now we need to apply the shapes to the main mesh and link them with the weightmaps. To do this, change to Shape Modeling mode, select your base mesh, and choose Deform\Shape\Select Shape key, and again choose your base mesh. This creates a default mesh shape from which the others are calculated. Choose Deform\Shape\Save Shape key, select the wide shape, and call it l_wide; repeat again selecting the same shape and call it r_wide. Under the main mesh, you will now have clusters for the shapes and weightmaps, as in **Image 5**.

Locate your base mesh in the explorer, and expand the shape cluster to show the l_wide and r_wide shapekeys. Select the l_wide shapekey and choose Deform\Shape\Connect with weightmap and scroll down to the weightmapcls cluster in your base mesh, expand it, and select the l_weight_map. Repeat for the r_shape. Open the mixer for the model and create two new shape tracks, and then import both l_wide and r_wide into the tracks. By increasing or decreasing the slider strengths for each track, you can see the effect of blending the strengths of each shape to effectively create three different positions, l_wide full, r_wide full, or both full strength for the full wide shape across the mouth, seen in **Image 6**.

Image 6– Inserting tracks and sources into the model mixer.



Once you have repeated this procedure for the narrow shapekeys, all that is left is to create a head bone and jawbone, for the mouth open/closed controls. The head bone obviously just connects at the neck joint; place the jawbone so that the rotation looks natural or normal. In the case of the round head of the example, it makes sense to place this joint near the center of the head. It's worth zeroing out the bones themselves, and although you can do this with a neutral pose, I prefer to use the 'Align Root to First Bone' option under the Animate/Skeleton menu, if possible. Next, change to the Animate mode and envelope the head. Weighting is important also, so spend some time ensuring a smooth transition from top lip to bottom at the sides of the mouth, as shown in **Image 7**.

SOFTIMAGE|XSI has several ways that you can create controllers to animate the shapes once they have been set up. One of the simplest is to create a custom parameter set containing parameters for each expression or visime. This doesn't give you much visual feedback though and also limits you to one visime per slider, which means you end up with a long list of parameters to search through for the one you want. I much prefer creating 3D sliders and parenting them underneath the camera, as you can link up to four shapes to them simultaneously, and because this is graphically a more understandable format.

First, create an implicit square, and make it 2 units big. Rename it 'jaw_square'. We will be writing expressions to link the mouth shapes to the controller and this will help to simplify the expressions. Create an implicit circle with a diameter of 0.2 units and parent it to the square without moving either of them. This will effectively zero out the circle's position and make it relative to the square. Translate the circle 1 unit up in Y (so it sits on the top edge of the square) and save a neutral pose.

Open the local kinematics page for the circle (Ctrl k) and select the Pos. limit tab. Enable pos. limits for X, Y, and Z, and set them up as follows;

Minimum: X limit -1, Y limit -2, Z limit 0
Maximum: X limit: 1, Y limit 0, Z limit 0

Check **Image 8** for an example. You should find that the circle is now limited to movement within the square. Rename the circle 'control'.

The next part is not an exact science, and depending on your setup, you will need to alter some of these numbers accordingly. I want to link the jawbone rotation to the control object, which has a range of movement in X from -1 to 1 and in Y from 0 to -2. Open the local kinematics page for the jawbone and right-click on the green divot for rotation in Y and choose Set Expression. The jawbone on my model rotates in Y from left to right, so the axis of the controller I want to link it to is the X-axis; model.control.kine.local.posx * -20, where Model is the name of the model null the head lives under and Control is the name of the circle. Right-click on the rotation in Z green divot and set Expression again, this time with the following: model.control.kine.local.posy * -20 and this should give you your jaw control as in **Image 9**.

To create a slider for the wide/narrow shapes, again create a 2 unit square and 0.2 circle, this time without setting a neutral pose, and rename the circle 'control2'.

I'll tackle the l_wide first, which affects the left of the character's face, or the right as we look at it. Open the model mixer and right-click on the animation divot for the shape, and choose 'set expression'.

So, for this shape, we want it to be full when the controller is at the top of the square, and fall off as the slider moves further left or down to the middle of the square. So we need to combine two conditional expressions.

cond(control2.kine.local.posy > 0, control2.kine.local.posy, 0)

This states that if the circle is greater than 0 in the Y- axis, the value will equal the value of the circle, and if less than 0, the value will be 0.

cond(control2.kine.local.posx > 0, 1, 1 + control2.kine.local.posx)

This states that if the circle is greater than 0 in the X-axis, the value will equal 1; if it is less than 0, then the value will equal 1 + the posx value, which is 0 at the center of the square and -1 at the left.

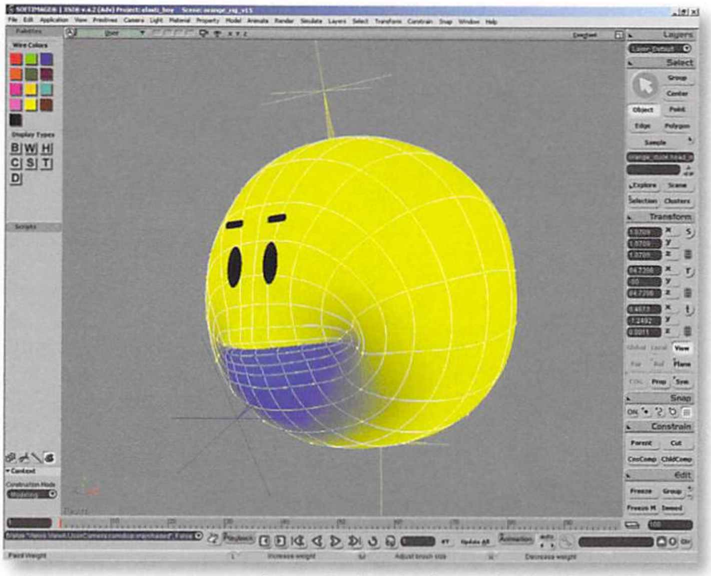


Image 7– Enveloping and weighting the model.

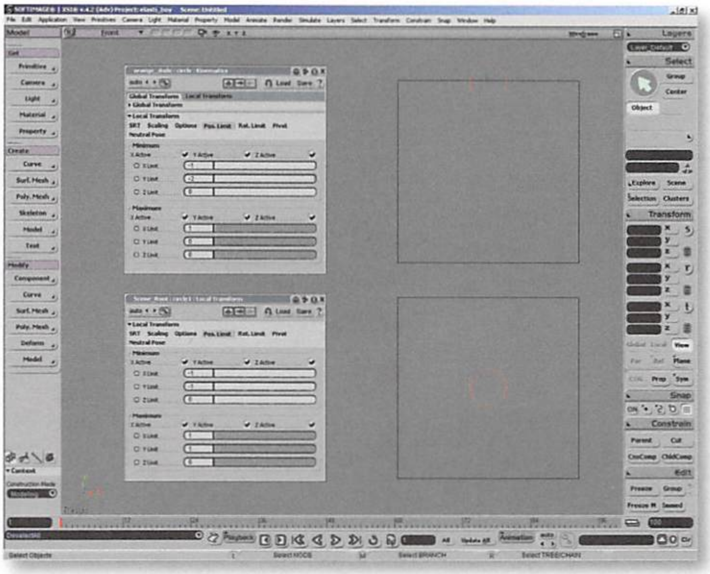
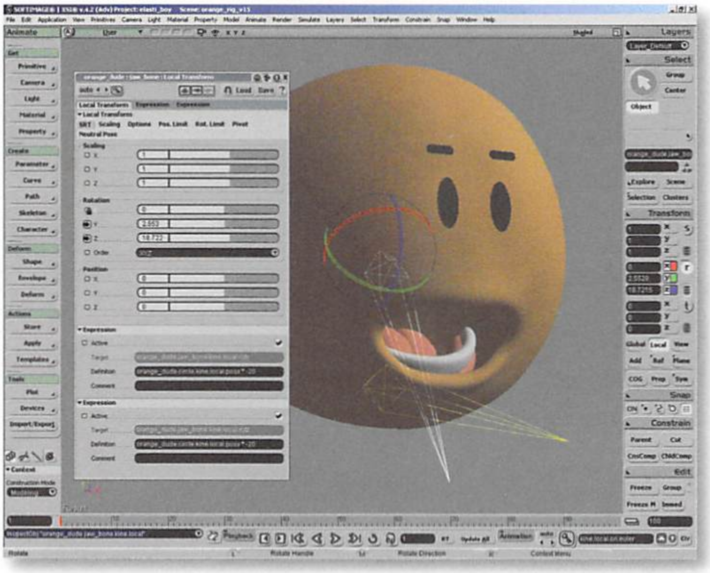
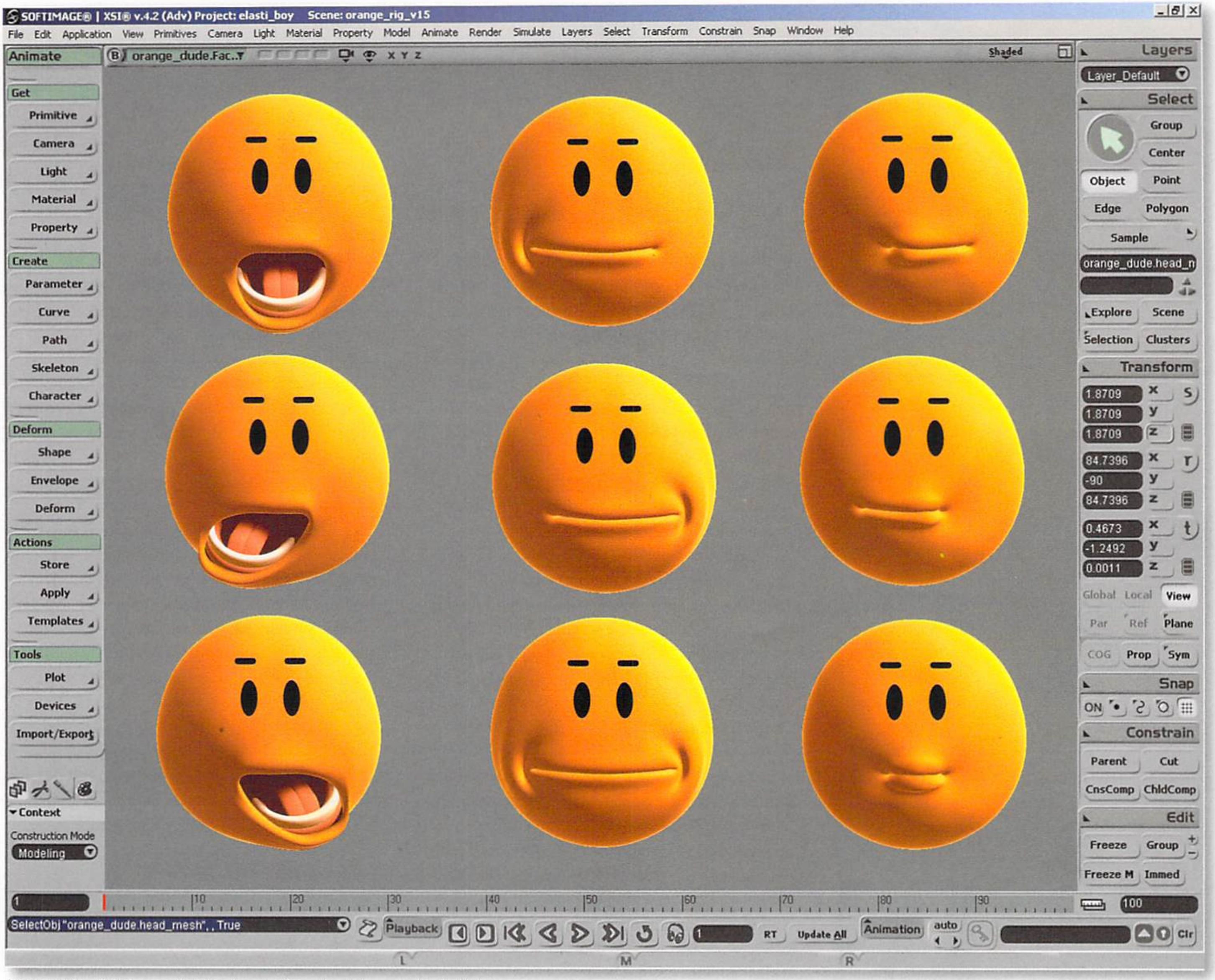


Image 8– Create two controllers for the mouth movements.

Image 9– Linking the jaw rotation to the controller.





SOFTIMAGE® | XSI® v.4.2 (Adv) Project: elasti_boy Scene: orange_rig_v15

SelectObj "orange_dude.head_mesh", , True

File Edit Application View Primitives Camera Light Material Property Model Animate Render Simulate Layers Select Transform Constrain Snap Window Help
Animate
orange_dude.Fac..
Shaded
Get
Primitive
Camera
Light
Material
Property
Create
Parameter
Curve
Path
Skeleton
Character
Deform
Shape
Envelope
Deform
Actions
Store
Apply
Templates
Tools
Plot
Devices
Import/Export
Context
Construction Mode
Modeling
Layers
Layer_Default
Select
Group
Center
Object
Point
Edge
Polygon
Sample
orange_dude.head_n
Explore
Scene
Selection
Clusters
Transform
1.8709
1.8709
1.8709
84.7396
-90
84.7396
0.4673
-1.2492
0.0011
Global Local View
Par Ref Plane
COG Prop Sym
Snap
ON
Constrain
Parent
Cut
CnsComp
ChldComp
Edit
Freeze
Group
Freeze M
Immed
100
Playback
RT
Update All
Animation
auto
Clr
L
M
R

Image 10– All the available shapes from just two control sliders.

To combine the two expressions, we simply multiply them together, because if they are both at max it is still only 1*1=1, so we are effectively normalizing the values.

cond(control2.kine.local.posy > 0, control2.kine.local.posy, 0) * cond(control2.kine.local.posx > 0, 1, 1 + control2.kine.local.posx)

The expression for the r_wide shape comes next, which is slightly different:

cond(control2.kine.local.posy > 0, control2.kine.local.posy, 0)

This part is the same as above, as you want the same effect on the Y-axis as the other smile.

cond(control2.kine.local.posx < 0, 1, 1 - control2.kine.local.posx)

On the X-axis, if the circle is less than 0, the value returned is 1; if it is greater than 0, the value is 1 - the posx value which is 0 in the center and 1 at the right, so it gradually cancels itself out the further right it gets.

Again, you combine the two expressions to get the following:

cond(control2.kine.local.posy > 0, control2.kine.local.posy, 0) * cond(control2.kine.local.posx < 0, 1, 1 - control2.kine.local.posx)

Continuing the same logic as above, the expressions for the narrow shapes (left and right) are as follows:

l_narrow:

cond(model.control2.kine.local.posy < 0, - model.control2.kine.local.posy , 0) * cond(model.control2.kine.local.posx > 0, 1, model.control2.kine.local.posx + 1)

r_narrow:

cond(model.control2.kine.local.posy < 0, - model.control2.kine.local.posy , 0) * cond(model.control2.kine.local.posx < 0, 1, 1 - model.control2.kine.local.posx)

So, there we have it; two controllers that between them will give you all of the shapes in **Image 10** – open, closed, open left, open right, wide left, wide right, wide both, narrow left, narrow right, and narrow both. Phew! The last thing to do is to create a camera and rename it face_cam or something similar, and parent the squares underneath the camera itself. To get the squares set up to face the camera, choose a square, select Match All Transforms, and select the camera. Translate the square locally in Z until it is a suitable distance from the camera and scale it without affecting the expressions.

Matt Morris *is an animator and rigger at Jellyfish Pictures in London and a co-founder of the XSI London User Group.*

FACIAL EXPRESSIONS

By Eric Keller
Scientific Animator
West Hollywood, California

Every once in a while a little piece of software comes along and people start raving about how it's going to change *everything*. I know, we've heard that one before. The more jaded of us just sort of shrug and say "oh really," confident that nothing could ever replace our beloved Maya, SOFTIMAGE|XSI, 3ds Max, or LightWave. I have to tell you though, after spending several months with Pixologic's ZBrush, I have become a believer – a raving, frothing at the mouth, diehard, cult joining, Kool-Aid drinking believer. ZBrush has already changed *everything*.

ZBrush is a great addition to your CG modeling and animation tool set. It works very well with the .obj format and thus fits as a compliment to all the major 3D modeling and animation software packages. In the past year, there has been an explosion in training tools for ZBrush, and the amount of fantastic artwork uploaded daily to the ZBrush central forum is astonishing. What is ZBrush? It's a 2 1/2 D Painting program that doubles as a modeling/texturing program – and, of course, much much more.

If you haven't tried it out yet, you really need to. The interface and the concepts behind how it works are a little daunting at first, but it clicks rather quickly. Most of the basics are covered in the zscripts that come with the software, and I'm not about to recreate those lessons here. This particular tutorial focuses on one way in which you can incorporate ZBrush into your Maya animation workflow. I'm only covering a very small amount of what ZBrush can do. Download the demo from the Pixologic Web site (www.pixologic.com) and play with it and you'll quickly appreciate how deep ZBrush is. I'm using a ZBrush 2.0/Maya 6.0 combination, but these ideas should work just as well with ZBrush and LightWave, 3ds Max, or SOFTIMAGE|XSI.

ZBRUSH AND BLENDSHAPES

Creating a full set of blendshape targets for the purpose of lip sync and facial animation is an arduous task. If done properly, it can take hours. I have found that the sculpting, masking, and editing tools in ZBrush really speeds this process up and makes it a bit more fun. Furthermore, when using ZBrush with a Wacom tablet, making blendshape targets is a downright joyful experience (which is what animation should be).

ZBrush is often used in conjunction with Maya to add detail or create displacement maps for existing models, or to create new 3D models from scratch. However, in this tutorial, I will take a polygon head that belongs to a character I originally modeled in Maya. I will use ZBrush to sculpt just a couple of example facial expressions on the model, which I will then bring back into Maya for the purpose of creating blendshape targets.

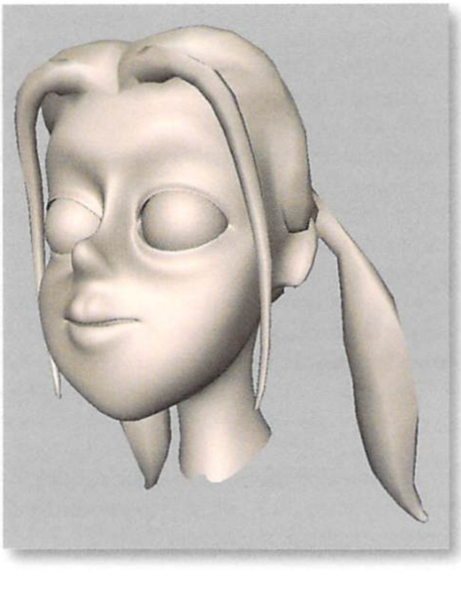
I'm not going to be subdividing the mesh at all; I'm just sticking to the same resolution that existed when I imported the model from Maya. I'm not adding any polygons or vertices to this model; as you know, when working with blendshapes, it is often necessary that all the blendshape targets maintain the same point order as the deformed model.

There are a few resources I always use when working on facial expressions. The first is a mirror, and the second two are books. *Stop Staring: Facial Modeling and Animation Done Right* by Jason Osipa is an indispensable book. Osipa covers the topic of facial animation much more deeply than I can in this tutorial, and some of his techniques form the basis of how I have set up the topology of my models and blendshape targets. The second book is *The Artist's Complete Guide to Facial Expressions* by Gary Faigin. It is an absolutely essential resource. In addition, I highly recommend the DVDs on ZBrush published by the Gnomon Workshop.

PART ONE: *THE MODEL*

1 I'm using the head from my surfer girl model as an example. Here she is in Maya, a simple polygon head weighing in at about 5645 polygon faces (see figure 1). She has some stand-in eyes and simple teeth that will be used as guides as I sculpt the facial expressions.

Figure 1 shows the original polygon head in Maya. Eyes, mouth, and teeth have been combined into one object.



2 To bring this in to ZBrush, I have combined the head geometry with the eyes and the teeth using *Polygons>Combine* under the modeling menu set in Maya. These parts of the model can be deleted later to make room for the actual eyes and teeth.

3 I export the model as a Wavefront object (.obj format) by selecting the mesh and then using *File>Export Selection*. In the options, I choose OBJexport with groups, Pack Groups, Materials, and Smoothing set to *Off* and Normals set to *On*.

4 I save the .obj object to a folder I can find easily from ZBrush, usually in the project directory I've created in Maya.

5 The next important task is to bring it into ZBrush. To do this, I start up ZBrush and then import the object as a tool by going in the tool menu and choosing Import. In the dialogue box, I find the .obj file I exported from Maya and select it.

By the way, models are called "tools" in ZBrush and appear in the tool palette. This may seem confusing at first, but it makes more sense once you've used ZBrush for a while. But, to be brief, everything you use to actually make stuff on the canvas is called a "tool." This means anything from a paintbrush to a primitive to an imported 3D model.

Furthermore, most of the parameters related to the model will be found on ZBrush's Tool Palette, which I'll be using a lot throughout this tutorial.

6 Once the object is imported, an icon that resembles the .obj file should appear in the currently selected tool window, but nothing will show up on the canvas until I draw it on.

7 At this point, I drag my cursor on the canvas (or the stylus across a Wacom tablet) and the object will appear and scale upwards.

8 Once I have dragged it to a decent size, I immediately hit the Edit button above the canvas to switch to Edit mode. If I don't do this, every time I drag on the screen, a copy of the Polygon Head tool will appear. This is very useful when you are creating a composition using a 3D object tool, but it won't help me in what I'm doing now.

9 Once in Edit mode, I drag on a blank part of the canvas; this will rotate the Head tool. I keep dragging until the head is close to facing me right side up, and then I hold the Shift key while dragging. This will snap it into an orthographic view. Now I'm ready to set the model up for the purpose of creating blendshape targets (see figure 2).

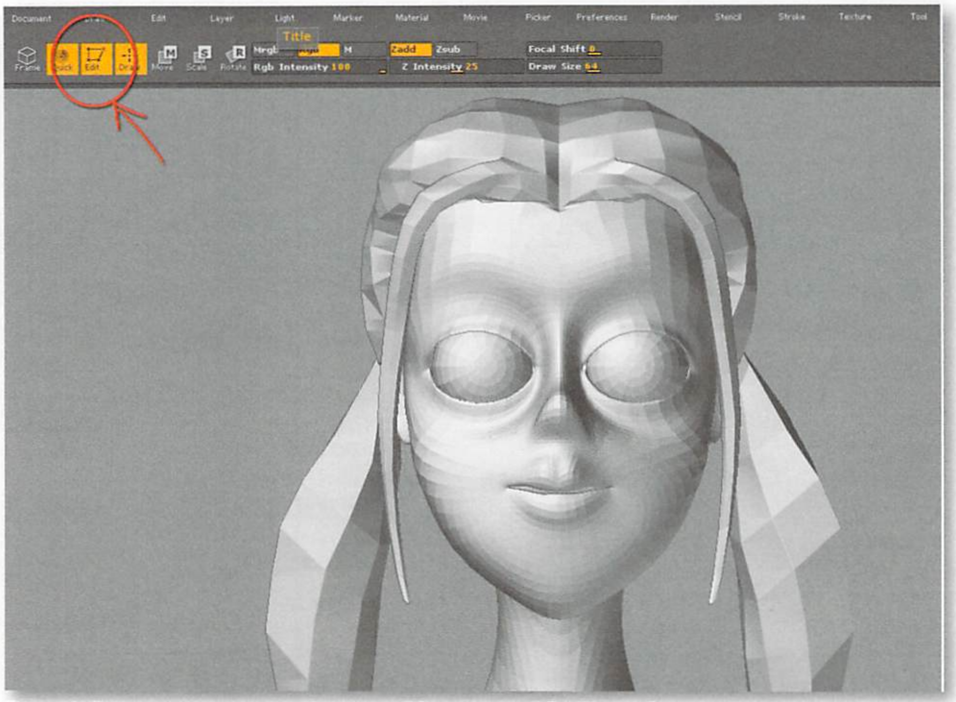


Figure 2–The model has been imported into ZBrush as an OBJ file. the Edit button is highlighted.

In this tutorial, I will be making two simple blendshape targets: a smile and a frown. I think these two simplified examples will be sufficient to demonstrate how ZBrush can be used to easily create blendshape targets.

PART TWO: *CREATING POLYGROUPS*

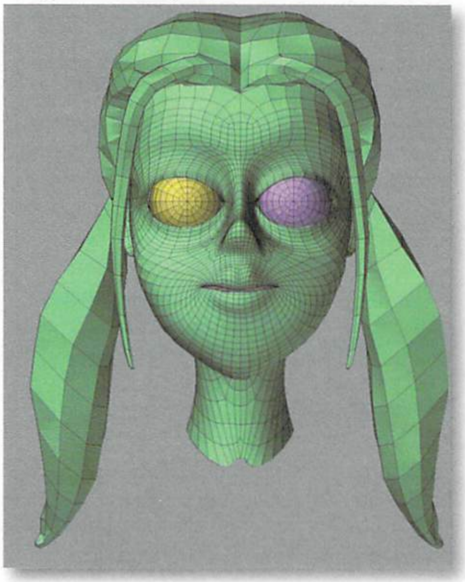
Polygroups in ZBrush is an extremely handy tool that is helpful in allowing me to divide and isolate sections of the face for easier editing. By using the polygroups feature, I can easily control visibility, masking, and selection. I can also save them with my model. Furthermore, I can easily recreate and reorganize them as many times as I want. If I am moving from a lower resolution model to a higher resolution version, the polygroups will persist and adapt to the other resolutions. I think of polygroups as a much more intuitive implementation of the selection sets in Maya; this is not exactly what they are, but for this tutorial that's a good way to think of them.

1 To create polygroups, I just need the cursor to select points and polygons and to paint masks, I really don't want to edit and change the mesh yet. So it is *very important* that I have the Z intensity set to zero when the Draw tool is activated. This way I won't accidentally draw a bulge or a dent on the face while I'm creating my polygroups. The Z intensity slider appears in the toolbox above the canvas in the standard window layout.

2 I make sure I'm in Edit mode and then I press the Frame button to display the wireframe on the mesh. (If the Frame button is not visible/grayed out, it's most likely because the Quick mode button is Off. Quick mode displays an unsmoothed version of the mesh to increase display performance.)

3 When I created the model in Maya, I used Combine to add the teeth and eyes to the model. Now that they are all in ZBrush, I can separate them into groups by pressing the Auto Groups button under the polygroups menu in the tool palette. This will divide any separated piece of the mesh into its own group. You can see the eyes and face change into different colors when the Frame button is On. Now I already have my first set of polygroups (see figure 3).

Figure 3–Pressing the Autogroups button separates the head, each eye, and teeth into their own polygroup. A different color has been assigned to each group.



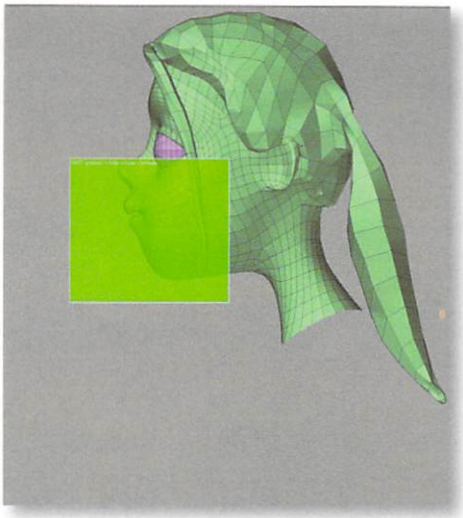


Figure 4–Holding Control and Shift while dragging creates a green box. Everything outside the box will be hidden when the mouse button is released.

There are two ways that I like to make my polygroups; I either Marquee Select and hide polygons or I paint a mask and convert the masked polygons into a polygroup. Allow me to expand on these two techniques.

First, I'll demonstrate the Marquee Select method.

4 On the far right side of the screen, I press the Pt Sel button. This allows me to more accurately select the individual polygons. With this on, selecting any point on the polygon is the same as selecting the whole polygon.

5 I rotate the model (in Edit mode you can do this by dragging the mouse or tablet stylus on a blank part of the canvas) to a side view, and when it is close to being a profile view, I press the Shift key while rotating so that it snaps to a perfect profile.

6 I hold down the Control and Shift keys and a green box appears as I drag the mouse around the mouth area that I want to isolate. By letting go of the mouse button *after* releasing the Control and Shift keys, the part of the model that is in the green box is isolated and the rest disappears. If I let go of the mouse *before* releasing Control-Shift, the box turns red and anything inside the red box is hidden (see figure 4).

The polygons that disappear have not been deleted, they are just invisible now. ZBrush temporarily pretends that these polygons no longer exist, which can boost performance on a dense model, but they can easily be brought back again by Control-Shift clicking on a blank part of the canvas. The analogous tool in Maya would be to select some polygons and use the *Show>Isolate Select>View* selected menu option in the view panel tools.

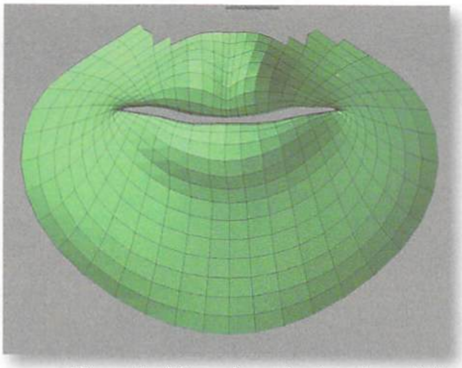


Figure 5–After a few minutes of carefully selecting and hiding polygons, the mouth area has been isolated.

However, I find ZBrush's implementation of this functionality to be much easier to use and more interactive.

7 At this point, I lightly tap with the mouse button on the Move button on the far right side of the screen so that the isolated mouth area fills the canvas. Tapping the mouse button on the Move, Scale, or Rotate buttons on the right of the screen will center the visible object and scale it to fit the canvas, kind of like the "f" hotkey in Maya.

8 I now use the Control-Shift mouse drag combination to start narrowing down the mouth area, making the polygons I don't want in the mouth polygroup invisible. This takes a few minutes of practice to get the hang of it. Experiment for a little bit with toggling the visibility of the mesh's polygons. Once again, if I need to restore visibility of the entire mesh at any time, I hold the Control-Shift keys and mouse click in an empty area.

9 The figure shows the mouth area isolated from the rest of the mesh (see figure 5). I spent some time rotating it around, Control-Shift dragging small squares around the vertices of the polygons until I ended up with this. Some parts of it were a little tricky since I have an interior section of the mouth as part of the mesh. Under Display Properties in the tool palette, I switched to Double-Sided so I could see the exterior face polygons as well as the interior mouth polygons. I also toggle the Local button on the far right so that the model rotates around the visible part of the mesh rather than the entire head geometry. The model's rotation updates to center around the most recently edited part of the model.

10 Once I have my mouth section properly isolated, I click on the Group Visible button under the polygroups menu on the tool palette. This makes the mouth geometry change color. By Control-Shift clicking on a blank part of the canvas, the rest of the head reappears and now the mouth section is a different color from the rest of the mesh (see figure 6). This saves me a great deal of time when I want to re-isolate this section. Now I can just Control-Shift click a vertex inside the colored mouth polygroup and the rest of the model disappears. Control-Shift clicking on the head geometry causes the mouth to disappear. If I save this head as a ZBrush tool ("Save as" in the tool palette), the polygroup will be saved with the model.

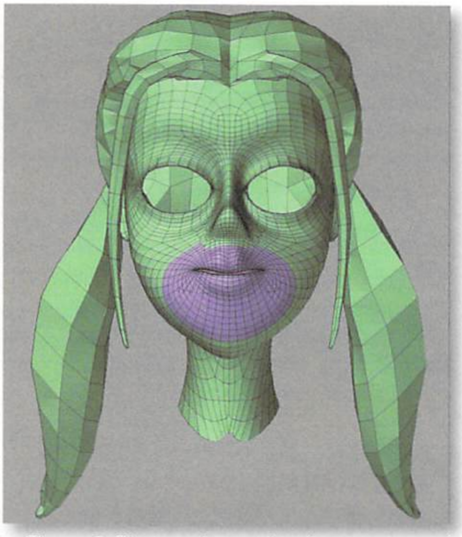


Figure 6–The mouth area has been converted into its own group and assigned a new color.

11 Now I go on to isolate the lower lip from the upper lip in order to create a polygroup for it. I use the same techniques to isolate the visibility of the lower lip.

12 Once the lower lip is the only thing visible, I click the Group Visible button again and the lower lip mesh changes color.

13 Now the head is one color, the upper lip is another, and the lower lip is another color. These colors correspond to the new polygroups. Every time you make a new polygroup, a new color is assigned to it automatically. If I Control-Shift click on a point that borders the upper lip group and the lower lip group, then the head disappears, but both lip groups remain. Control-Shift clicking on the upper or lower lip groups toggles the visibility as well. Control-Shift clicking on a vertex at the border of two groups hides everything but those two groups. It's a good idea to practice Control-Shift clicking on the various groups to get a handle on how to control their visibility.

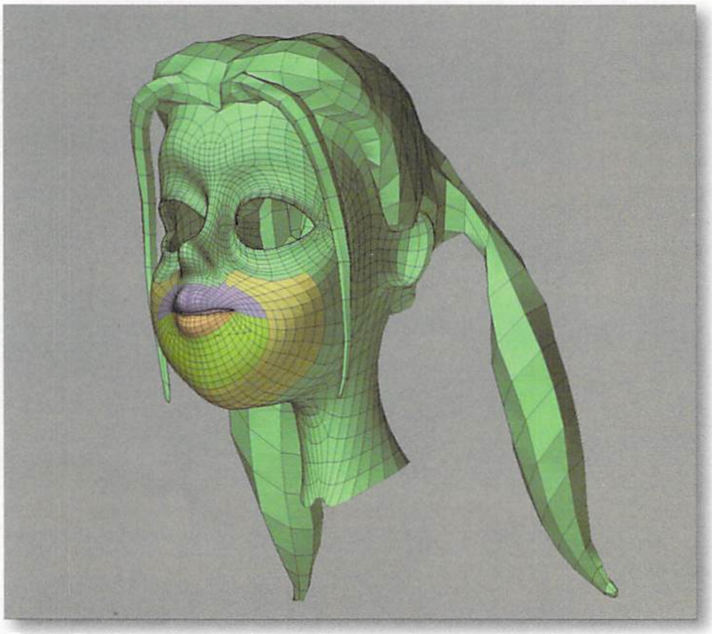


Figure 7–With a little bit of work the mouth area has been further subdivided into several polygroups.

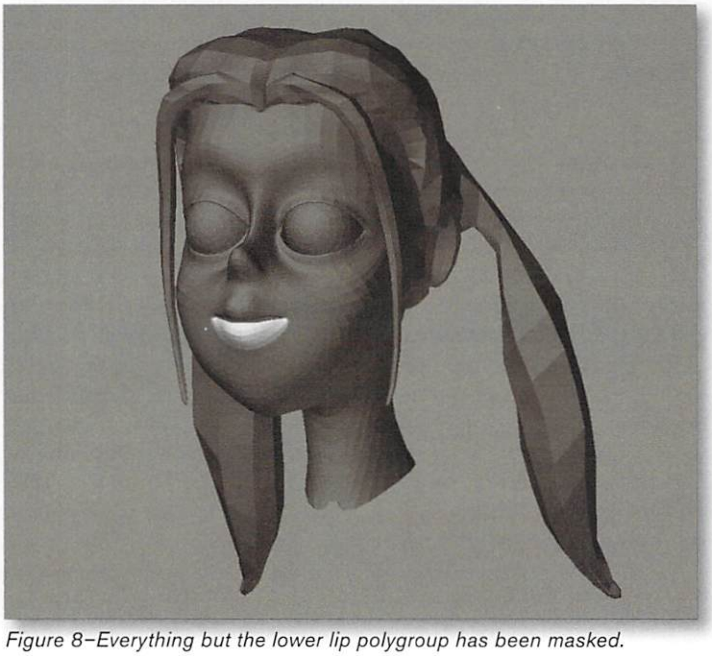


Figure 8–Everything but the lower lip polygroup has been masked.

When working on a model like this, sometimes I will go ahead and make polygroups for every part of the face I plan to work on, and other times I will just work on one part at a time and then erase the polygroups by making sure the whole mesh is visible and then pressing the Group Visible button under the polygroup section of the tool palette (see figure 7).

Furthermore, since the polygroups are saved with the model when it is saved as a tool, I often save several versions of the model with different arrangements of polygroups. Since these models will be used to create facial expressions, I can always create more blendshapes by loading up a version of this mesh with the appropriate polygroups, and then edit and export as necessary.

MASKING

The reason I go to such lengths to create polygroups becomes more obvious when I start using masking to edit the model. Masking allows for part of the model to be edited while the rest remains intact. For instance, if I want to edit just the lower lip area, I use these simple steps:

1 I Control-Shift click on a vertex in the lower lip polygroup, making it the only thing visible.

2 Under the masking menu on the tool palette, I click Mask All.

3 I Control-Shift click on the canvas to make the mesh visible again. Now just the lower lip area is masked.

4 Under the masking menu, I click Inverse, and the mask is now applied to everything but the mouth.

5 Now I edit away, and if I need to, I can swap the masked and unmasked parts by hitting the Inverse button.

6 If the polygroups and wireframes are distracting me while I'm editing, I turn off the Frame button and this changes to a view of the model without the wireframe. The polygroups are still there, they are just not visible while the Frame button is off. Likewise, I can toggle the visibility of the mask by pressing the View Mask button under the masking menu on the tool palette (see figure 8).

PAINTING POLYGROUPS

If you find that the Control-Shift click, select, and hide method gets a little tedious or that you want to really refine your polygroups on a higher resolution model, you can try painting a mask on the model and then converting the mask into a polygroup. Here's how I do it:

1 To see the mask more clearly, I turn off the Frame button and make sure the material assigned to the model is something simple like a fast shader. I then make sure that the model is a light color by choosing something like white from the Color Selector.

2 I make sure I am in Edit mode and that I have the Draw button on.

3 While holding the Control key down, I paint a mask on the surface of the model; it should appear as a gray blotch on the surface. If it is not showing up or if it is very light, it may mean that the RGB intensity is too low. That's okay sometimes if you want to paint a mask with a lower masking intensity. However, for this particular purpose, it is best to have a nice strong mask with the RGB intensity at 100. The RGB Intensity slider is located in the bar above the canvas.

4 Once I have the mask painted the way I want it, I hit the HidePt button under the masking menu in the tool palette. This hides all of the geometry that has a masking value of less than 50% (this is why I wanted to make sure my RGB intensity was at 100).

5 Once again, with the Frame button on, I hit the Group Visible button under the polygroups menu and the visible chunk I created from painting the mask changes color. When the entire model is visible, it is possible to see the polygroup I've painted.

6 Now I can continue as before, masking these polygroups and editing to create my blendshapes.

CREATING SHAPES

The first blendshape target I want to make is a smile. Smiles are tricky, partly because we have such a hard-wired misconception of how a smile should look. A smile is not just a matter of mouth corners going up. It's actually the effect of the mouth going wide, curving around the front of the teeth, and with the corners going back towards the nasolabial fold of the cheeks. The lips tend to get thinner as they are being stretched as well. A truly convincing smile will cause the bottoms of the eyelids to flatten and move up. However, that's a matter for a separate blendshape target. For this target, I will only be editing the mouth area. Without the lower eyelids going up the smile will look a little phony and slightly weird, but that's OK for what I need to make right now.

1 I load up my base model and create the polygroups as described above so that I can easily isolate the mouth.

2 Before I start moving things around, I click the StoreMT button under the Morph Target menu in the tool palette. This will store the base as a morph target and allow me to compare the version I'm working on against the original base mesh whenever I hit the Switch button.

3 Now I use the polygroups I've set up to mask just the mouth area, just like I described in the previous section.

4 I start by making my brush size fairly large, with a large focal shift by adjusting the Draw size and the Focal Shift sliders in the upper right. Focal Shift is essentially the falloff area of the brush.

5 I switch my editing tool from Draw to Move by clicking the buttons above the canvas.

Incidentally, the Move tool on the *right* of the canvas actually moves the object around the canvas, the Move tool *above* the canvas allows me to move vertices while in Edit mode. A little confusing, but it makes sense after you play with it a little. If you forget what a button does, you can always move the mouse over the button and press the Control key. A little explanatory paragraph should pop up that will describe what the tool does.

6 One more thing before I start pulling and pushing – I turn on symmetry across the X-axis by clicking the >X< button on the transform palette. I also hit the >M< button so that the edits are mirrored across the X-axis. This tool is similar to LightWave's symmetry mode, but ZBrush's implementation can get very sophisticated once you use symmetry across multiple axes or when you start using the radial symmetry mode.

7 Now I'm all ready to go. I start by carefully moving the corners of the mouth a little up and to towards the cheeks. I stretch the mouth wide and pull those corners back. I notice creases forming at the edge of the masked area; that's OK, I can easily smooth those creases when I've got the smile a little further along. It's very important to check from the side to make sure the mouth corners are going back a little towards the ears and not just up or out to the sides. This is why I tend to make very small adjustments; it's better to do a little at a time (see figure 9).

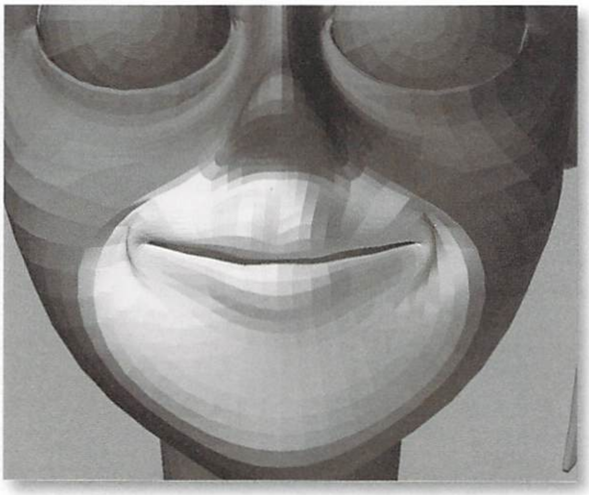


Figure 9–Using the Move Tool and symmetry across the X-axis, we have the beginnings of a smile.

8 It's also important to be wary of moving the vertices close to the center of the face. Sometimes, even with symmetry on, it's easy to accidentally move the center vertices out of alignment, causing a slightly lopsided smile. Sometimes this can be fixed by using the Smart Resym tool under the deformation menu in the tool palette. Occasionally, I will mask out the polygons at the very center of the model to prevent this from happening.

9 Once I have roughed out how where I want the corners of the mouth to go using the Move tool, I switch to the Draw tools to start refining the smile. To do this, I click on the Draw button next to the Move button above the canvas. These buttons are also located on the transform palette.

10 The Draw tool has several modes, which are very useful in refining the model. These are buttons located on the transform palette and are labeled Std (Standard), StdDot (Standard Dot), Inflat (Inflate), InflatDot (Inflate Dot), Layer, Pinch, Nudge, and Smooth. The tools' parameters and effect on the model can be adjusted by playing with the eit curve and the Z intensity. Switching from Z add (on the menu above the canvas) to Zsub essentially reverses what the tool does. Reversing smooth does not have any affect though. Detailed explanations of each of these tools are available in the ZBrush documentation. I have my preferred tools and settings. Play around with them a bit on an object and you'll develop a sense of how you can use them.

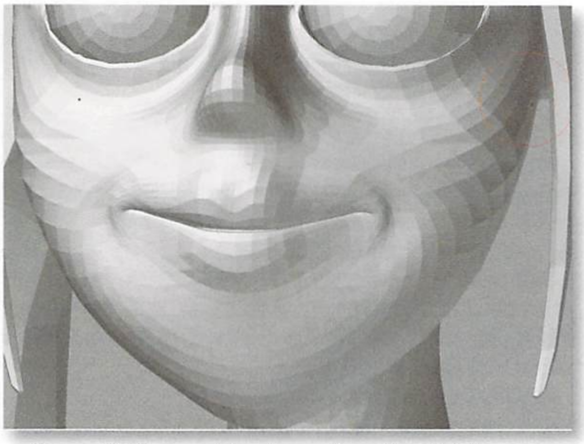


Figure 10–The mask has been cleared so that the creases at the mouth corners can be refined and smoothed.

11 I use the Draw tool in smooth mode with a low Z intensity to smooth the corners of the mouth near where it meets the edge of the mask. I want to have a crease there so I don't smooth it completely. At this point, I switch modes fairly frequently, using Inflate to close up the lips where there may be gaps. I use the Pinch mode for this function as well. I use the Nudge mode to refine the position of the mouth corners and Pinch again to flatten out the fleshy parts of the lips. I use the Standard mode to raise a little skin near the mouth corners just outside the lips. Holding the Shift key while drawing can toggle on the Smooth mode; this can be very handy when you start switching quickly.

12 As things progress, I deactivate the mask by hitting the clear button on the masking menu on the tool palette. I can always get that mask back because of the polygroups I have set up. If I just created masks by painting them when I need them I'd find myself doing the same work over and over and it could start to get tedious and sloppy.

13 Next, I smooth out the creases at the mask borders. I use the Standard and Inflate modes to bring out more of the crease at the nasolabial fold and to puff up the cheeks to show that the mouth corners are actually pushing into the flesh. Finally, I may flatten out the end of the nose a teeny tiny bit for a subtle effect. Some of the skin on the chin can be moved up ever so slightly using the Nudge mode (see figure 10).

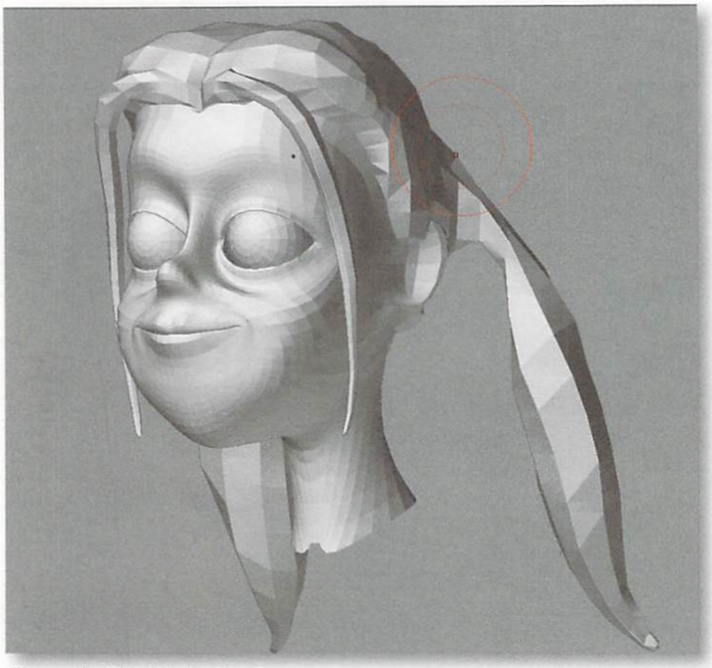


Figure 11–The finished smile.

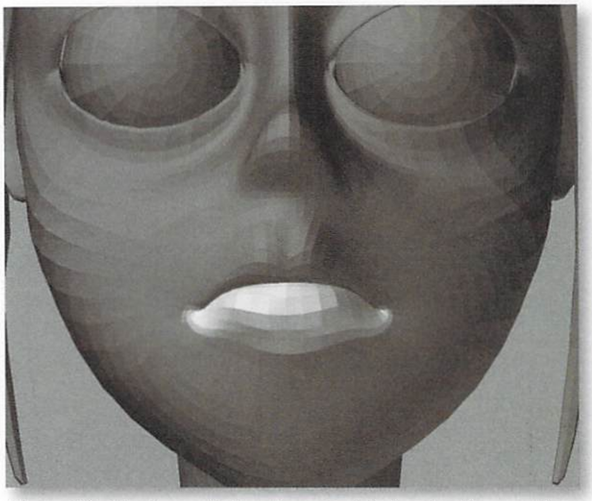


Figure 12–The lower lip has been isolated using the masking tools so that it can be modeled into a pout.

14 I switch back to the base shape frequently to compare the smile with the original by using the switch button under the Morph Target menu on the tool palette.

15 Once I'm happy with the smile, I delete the base morph target using the DelMT button on the Morph Target menu, and then I save the tool in case I want to edit it in the future. Under the tool palette (not the document palette!), I choose "Save as," and then I name the tool "smile_1.ztl." If I need to bring it into ZBrush again later, I can use the Load button on the tool palette. The tool will load with all of the polygroups I have set up still in tact (see figure 11).

16 After saving the tool, I export the tool as an .obj file and put it in a folder in my Maya project files. Eventually, I will load this object into Maya and use it as a blendshape target. When I bring the object into Maya, I don't want the mesh broken up into polygroups, so I make sure the Grp button in the Export menu on the tool palette is deselected before I export a model. There may be occasions where having the groups in Maya is useful, but not in this particular case.

NOW FOR THE FROWN SHAPE...

The frown is similar to the smile in that the driving force of the expression is in the mouth corners. However, the mouth does not go nearly as wide as it does in the smile and the mouth corners' vertical movement is slightly more dramatic than in the smile. Also, the frowning action can push the lower lip out a little more into a pout. The upper lip will flatten as the corners drag it around the fleshy part of the lower lip.

1 First I clear the screen. This can be done by switching out of Edit mode and hitting the clear button under the layer palette.

2 I load my original Base Mesh tool into ZBrush and switch to Edit mode.

3 I store a morph target for later comparisons by clicking on the StoreMT button under the Morph Target menu on the tool palette.

4 I switch to Quick mode and turn on Frame so I can make sure that my polygroups are still there from when I set up this tool. I should have a separate polygroup for the upper and lower lips.

5 I use the polygroups to mask off the mouth section by Ctrl-Shift clicking on a vertex that is on the border of the upper lip – the lower lip and the chin polygroups. Everything but the mouth area should disappear (this may take a few tries before hitting the right vertex).

6 Just like before, I then hit the Mask All button under the masking menu on the tool palette. I Ctrl-Shift click on a blank part of the canvas and the rest of the head appears, but the mouth is masked.

7 I invert the mask so that everything but the mouth area is masked and now I'm ready to start editing. It sounds like a long way to go to get to the editing, but once you get the hang of it, it starts to go very quickly and naturally.

8 I switch to the Move tool (the button above the canvas, not the one on the right of the canvas); make sure that symmetry across the X-axis is On. Then, I carefully move the mouth corners down. I may move the entire mouth down very slightly as well.

9 Now I use the polygroups I have painstakingly set up to isolate the lower lip. I use Inflate to puff it up and the Nudge tool to bring the inside part of the lower lip out a little to give the effect of the lower lip rolling outwards into a pout (see figure 12).

10 With the Draw tool selected, I use the Smooth, Pinch, and Inflate modes to close up the lips and flatten the upper lip. I nudge some of the vertices of the upper lip out slightly towards the corners to give the effect of the upper lip spreading around the lower lip.

11 I rotate the model to a side view so I can check to see that the lower lip is pushing out the way I want.

12 I turn off Quick mode to see a more smoothed version of the model. This may require adjusting Dsmooth in the display properties on the tool palette to something like 0.1. *Note that I am not subdividing the model! I'm only adjusting how it is displayed.*

13 Additional touches include puffing out the skin outside of the mouth corners and pushing the flesh on the chin up into to bottom of the lower lip and out a little. The nose tip could flatten a tiny bit and the edges of the nostrils could come down a little, but that might be too much detail.

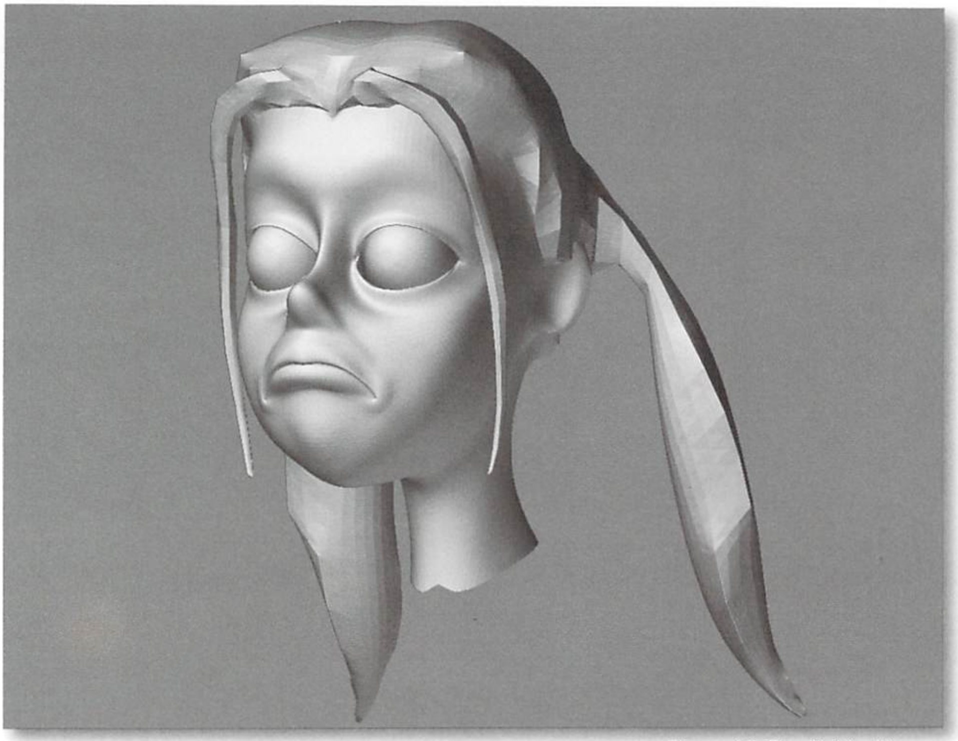


Figure 13–The finished frown.

14 Once I'm happy with the frown, I save it as a tool from the tool palette with the name "frown_1.ztl" and I export it as an .obj file (see figure 13).

SETTING UP BLENDSHAPES IN MAYA

Converting these objects into blendshapes in Maya is extremely straightforward.

1 I import my three .obj files into a Maya scene. I have my base object, my smile object, and my frown object.

2 I rename my three objects base, smile, and frown accordingly.

3 I select my smile and frown shapes first and my base shape last, and in the Animation menu set, I choose *Deform>Create Blendshape*. In the options, I name the blendshape "smile_frown" and I keep the origin local so the head doesn't move if my blendshapes target models translated anywhere in the scene. I leave Check Topology on.

4 Once I have created the blendshape deform, I can hide, move, or even delete the smile and frown blendshape target objects.

5 I then open up the blendshape controller by choosing *Window>Animation Editors>Blend Shape...*

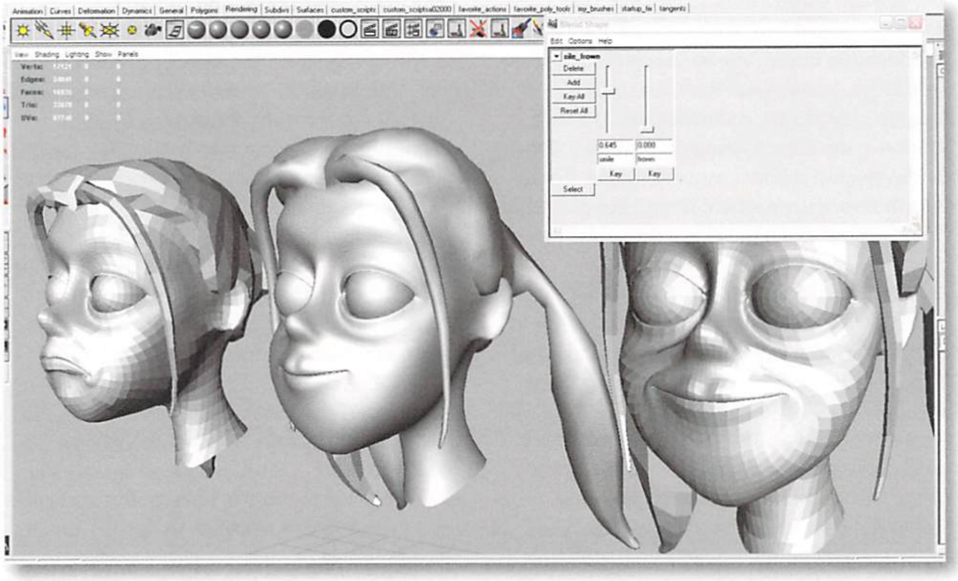
6 Animating my character's very bipolar expression set is as easy as moving the smile and frown sliders up and down and setting keys.

7 Finally, to lose the chunky look of my character's face, I select the base object, and in the modeling menu set, I choose *Edit polygons>Normals>Soften/Harden>All Soft* (see figure 14).

So now I have a couple expressions for my character and I can easily create many more using this same workflow. As in many animation tasks, a lot of work goes into the initial set up so that the actual task of animation is flexible and intuitive. The ZBrush/Maya workflow is becoming more established in the industry and as you can see, the techniques could easily be converted to a ZBrush/LightWave or SOFTIMAGE|XSI or Max workflow. This demonstration is only a tiny peek at the capabilities of Pixologic's ZBrush. I intentionally focused on one specific task to keep this article to a reasonable size, but there is much more power available in ZBrush for you to discover.

Eric S. Keller *is a digital artist and animator living and working in West Hollywood, CA. He enjoys learning, studying, teaching, creating, and spending time with his wife and their two dogs. His background is in art, music, and science.*

Figure 14–The models imported back into Maya and converted into blendshape targets. The model in the center is the base mesh, halfway into a smile. I'm pleased as well, actually.



smile_frown

smile 0.645

frown 0.000

ResPower Super/Farm™ Self-Service Render Farm 24x7 Access

Come See Us At SIGGRAPH: Booth #2102

TeraHertz At Your Fingertips

www.respower.com 866-737-7697 / 256-533-1090

Enjoy more beach time with a system from T.S.!
T S Computers
Ask about our SIGGRAPH Specials
SERIOUS MAGIC
NewTek
PINNACLE SYSTEMS
T.S. Computers 11300 Hartland Street North Hollywood, CA 91605 818-760-4445 www.vgn.com

Introduction to mental ray Programming

By Alan Jones
Architectural Visualization Animator
London, England

Much of mental ray's strength comes from its flexibility. Customizing it to provide additional tools is much simpler than most other raytracers. This is considered the domain of specialists, which is true for the most part, but in this article I hope to give an overview of how even those with only limited coding skill can leverage some of this power to create basic tools to enhance their pipeline. I'll do this through the example of a reflection shader with distance-based falloff.

We won't cover vendor-specific requirements for shaders here, as the method for installing a custom shader varies between software. For instance, SOFTIMAGE|XSI requires an spdl file be created and installed, while Maya needs a .mi file created, a config file edited, and the shader library and .mi file placed in a particular location. The sdk documentation of your program should give you details on its specifics. I've provided some resource files for your download at *http://www.hdri3d.com/resources*.

Before we start thinking about coding, let's just think of what we require our shader to do. At the simplest level, we want to cast a reflection ray and alter how strong the reflection is based on how far away the reflected object is.

Let's expand this a little. First, let's discuss casting a reflection ray. When do we want to? On any area of the surface that is reflective. Now, to expand altering based on how far away the reflected object is. Instead of fading to black over a distance, we'll use a near and far distance to control the range. Then, to control fading, we'll use two colors for reflectivity – a near and a far color. This gives the user a bit more control.

We need to keep expanding on our definitions of the steps until we end up with small enough pieces that we can easily manage them in our code. We know when we're going to cast a reflection ray – now let's look at how. That involves two steps: getting the direction we want to cast the ray, and then casting it. We'll expand that once more to allow for ray depth settings (we'll cover this more later, but those of you who've used mental ray will already be familiar with the term). So, we'll get the ray direction and check the ray depth; if we're within it, cast a reflection ray, if we're not, cast an environment ray. This way, we've can fall back on environment reflections rather than just returning black when we exhaust our ray depth.

Now to controlling the reflectivity based on distance. First, we need to determine the distance away the object was. Next, find out how far along between the near and the far distance this is. We'll express it in the range 0-1, where 0 is near and 1 is far. Then we'll mix between the near and far reflectivity using this number. To help prevent user error, we'll check if the far distance is closer than the near distance. If it is, we'll switch the far distance and color with the near distance and color.

Lastly, we'll add one more piece. Take a surface color and composite it with the reflection. Also two choices for the compositing mode could be useful - mix and add. Add will simply add the reflection on top of the surface color. Mix will reduce how much we see of the surface (multiply it by the inverse of the near weight) before adding the reflection.

From this information, we can put together a list of the parameters we'll need from the user. We'll need a number of colors – surface, nearReflectivity, and farReflectivity. The near and far distance should be scalars (numbers which can have decimal places). Finally, the Mix mode; while it's only one of two options (making a Boolean ideal), we'll use an integer so that you can expand the shader later by adding more compositing options.

Here's an overview of all the steps we've come up with. We'll put this in our code as comments before we start writing to give ourselves a guideline to work to.

- Check if either near or far reflectivity are not black. If they are both black, then we'll just get the surface color and return it.
- Get the direction for the reflection ray.
- Check if we've reached maximum ray depth: if we have, cast an environment ray, and if we haven't, cast a reflection ray.
- Check if the far distance is less than the near distance. Switch far settings with near settings if it is.
- Calculate our blend amount from the distance. If we decided to cast an environment ray, or if we didn't hit anything, we'll set the blend to 1, as we want both these scenarios to count as being far away. Otherwise, we'll calculate the blend as the % between near and far that the distance lies.
- Use the blend amount to mix between the near reflectivity and the far reflectivity.
- Calculate the reflected color by multiplying the color our reflection (or environment) ray returned by the reflectivity.
- Get the surface color.
- Composite the surface color with the reflected color using whichever mixing mode.

Now, as you can see, you've gotten all this way without coding a line. You've got all the main steps lined up in front of you, ready to go. I'm sure many of you are thinking this is a ridiculous example because you already know how to code shaders. This is half-true; my experience really allows me to speed up this process. I've been able to create a slightly more detailed coverage of the steps than someone without shader coding knowledge may have. When writing your own shader, don't be scared off in the planning phase. Write down the steps you have the knowledge for, dividing it up as much as possible. Once you've hit your limit, start coding. Write all the pieces you can translate to code. Then, get the information you require to cut the other pieces up until you know all the information you need to complete you shader. I go through the planning phase with every shader, regardless of the complexity, because it forces you to overcome many of the issues you'll face before you've spent any time on coding. This also means you may realize something that would have otherwise forced you to throw away hours, or even days, worth of work.

It's time to start talking about mental ray. mental ray shaders are C libraries; however, you can (and I do) code them using C++. They require two functions: one named after your shader and another named shadername_version. The shadername function contains all calculations performed when each shader is performing the bulk of its work (during raytracing for most shaders). The shadername_version function just needs to be there for mental ray. Just have it return 1 to prevent any problems. Optionally, you can also have a shadername_init and shadername_exit function. Use these for performing tasks (such as data initialization and cleanup) that only need to be performed once for the shader.

The shadername function is passed three variables: a pointer to a color, which stores the color resulting from your shader; a pointer to a state, (we'll cover this in more detail soon); and a pointer to struct containing parameters.

There are a few terms we should cover quickly before we go into details. The first is pointer. This is a C/C++ term. Unlike a variable, which stores its information in memory, a pointer stores a memory address that contains the information. I won't go into the details, as it's more a subject for a C/C++ book, but here's what you need to know about it for our purposes.

If you have a variable, you access its value directly. For example, use i=1;with an integer i. Use *i = 1; if you have a pointer to an integer and want to change the value. The same applies if you're giving it's value to another integer j. Assigning variable i to j, use j = i;. If i is a pointer and j isn't,, use j = *i;. Finally, if both are pointers, use *j = *i; to copy the value of i to the value of j.

The next term you'll need to know is a struct. A struct is a variable made up of other variables. If you made a struct type called shader_params, for instance, then you would make a variable of that type using shader_params variablename;. The way you access these parameters depends upon whether you have a pointer to a struct or just the struct. If our param_struct has a parameter called surface, access it with variablename.surface. If it is a pointer, we use variablename->surface.

Finally, some functions may want to be given pointers, but you want to pass them your variable, which isn't a pointer. The solution to this is to prefix your variable with an &. So, myFunction(&myVariable); would pass myFunction a pointer to myVariable. If myVariable was a pointer already, you wouldn't need the &.

Back to the state pointer we received. A state is a struct mental ray has defined. It contains information about, shockingly enough, the state of mental ray. Things like the current surface hit by a ray and the normal of the surface point hit, as well as more general information about the render, like resolution; there is a list of them in the mental ray documentation. If you're more adventurous, search for shader.h on your hard drive and have a look to see the C definition of mental ray structs.

Now to our shader. First, let's create our header file. This is the file that declares the functions and structs we plan to use in our shader. I've called the shader depthMirror, so our header file will be depthMirror.h and the C++ file depthMirror.cpp. You can see the contents of depthMirror.h in image **depthMirror.cpp** (in the downloaded resource files).

As you can see, our struct has its parameters listed by type, then name, followed by a semicolon. A struct can also contain structs (for instance, miColor is a struct made up of four miScalars – one for each of the RGBA channels), but if you're putting your own structs within another, ensure the struct is defined above the struct that includes it.

I won't cover full details on C/C++ syntax. If you follow the example, you should be able to create your own easily enough. Should you wish to gain a greater understanding of C++ syntax, I'd recommend Bruce Eckel's books Thinking in C++ volumes 1 and 2, available in electronic format free from *http://mindview.net (http://mindview.net/Books/TICPP/ThinkingInCPP2e.html)*

We don't require depthMirror_init or depthMirror_exit in this shader, so let's go straight into implementing each of the items we've listed.

We check to see if our reflectivity values are black so we won't waste time calculating ray directions and casting them, only to discard the information. This is one of the most basic methods you can use for optimizing your code, but also one of the most effective. First, we need to get the values for nearReflectivity and farReflectivity. To read in the parameters in mental ray, we use one of the mi_eval_* functions. In this case both of them are colors so we'll use mi_eval_color. It doesn't matter to us whether the user has put a color directly into the parameter or another shader. The mi_eval_* functions take care of all this for us. So, we declare a variable for each color and read it in.

```
miColor nearReflectivity = *mi_eval_color(&in_pParams->m_nearReflectivity);
```

If you remember what we said about pointers, that line should make sense. Our nearReflectivity variable is not a pointer, but the mi_eval_* functions return pointers. So we use the * in front of the mi_eval_color to assign the value to nearReflectivity. You'll also notice the & in front of in_pParams->m_nearReflectivity. This is because the mi_eval_* functions want a pointer to the color, but m_nearReflectivity is not a pointer. You can give a pointer to a variable by prefixing it with &. The -> is used because in_pParams is a pointer. I realize these concepts are a bit daunting, but hopefully I've explained it well enough to get you by.

Once you've got both your variable in we should check if they're black. I've split this off into a function in this case to tidy our code a little. Because the function is so simple, I've made it inline, – which means it's compiled as though written fully within your code. For simple functions like this one it's more efficient than it being a fully-fledged separate function. So all we'll do is check if both are black with

```
if (isBlack(nearReflectivity) && isBlack(farReflectivity)) {
```

The && means that both of the functions have to return true. Now to our isBlack function. It is a simple single line.

```
return (!color.r && !color.g && !color.b);
```

Because a value of 0 is considered false, we can use this to do our check. The ! means not. So when that is evaluated, a value of 0 would return true. If all the RGB channel return true, then it's black. We do this rather than checking if each is equal to 0 because it's faster to execute.

Now, if it is true, then we'll set the output color to our surface color and then return miTRUE.

```
*out_pResult = *mi_eval_color(&in_pParams->m_surface);
return miTRUE;
```

When a function returns, it doesn't go any further through its code. Returning finishes execution and the function decides its job is done. So, as you can see, when there will be no reflections, we've skipped all the work.

Getting the reflection direction is much easier than you might think. In fact, we don't need to do any of the math for calculating ourselves at all. mental ray has a built-in function, mi_reflection_dir, which calculates it for us. We will just create a vector for the function to store it in and then call the function.

```
miVector reflDir;
mi_reflection_dir(&reflDir, state);
```

As you can, see mi_reflection_dir needs a pointer to a vector and the current state. Because we've given it a pointer, when it changes the value, the change will stay in our variable. If it just took a variable, we wouldn't get the new value. It uses the state to get all the information it needs to calculate the new direction.

To calculate whether we've reached the maximum ray depth, we'll need to access some information from the state ourselves. state->reflection_level tells us the current reflection depth and state->options->reflection_depth tells us the maximum reflection depth the user set. Based on this information,, the other variables are self-explanatory. Here's the line we use.

```
bool traceRefl = state->reflection_level < state->options-
>reflection_depth && state->reflection_level + state-
>refraction_level < state->options->trace_depth;
```

Now we have a variable that tells us whether we should cast a reflection ray or an environment ray. Both functions require a pointer to a color, as well as the state and ray direction.

```
miColor reflColor;
if (traceRefl) {
        mi_trace_reflection(&reflColor, state, &reflDir);
} else {
        mi_trace_environment(&reflColor, state, &reflDir);
}
```

The color of the reflection is now stored in reflColor. Our next step is to check our near and far distance and switch them if necessary. I won't put in the mi_eval_* functions – you can see the full source in image X.

```
if (farDist < nearDist) {
        miScalar tempDist = nearDist;
        miColor tempRefl = nearReflectivity;
        nearDist = farDist;
        nearReflectivity = farReflectivity;
        farDist = tempDist;
        farReflectivity = tempRefl;
}
```

To calculate the blend amount between nearReflection and farReflection, we need to get the percentage position of the reflection ray's length. A ray gets its own state, a lot like the one we accessing, but with its own details. Because we cast the ray ourselves, we can access its state through our state's child, which is in state->child. The length a ray has traveled is in its state's dist property. So, the length the reflection ray traveled is in state->child->dist. But what happens if we didn't hit anything? In this scenario, the reflection ray would return the same result as the environment ray, so we still have a color we want to use for the reflection. When this happens, though, the dist property is 0 or less. We'll combine this with our traceRefl variable we created earlier to decide what to do.

```
miScalar blend;
if (traceRefl && state->child->dist > 0) {
        blend = getBlend(state->child->dist, nearDist, farDist);
} else {
        blend = 1;
}
```

By checking the traceRefl first, we may avoid doing a comparison lookup on the dist when we are at the maximum ray depth. As you can see, we've called a getBlend function. This is one we've defined ourselves. It takes the three distances and calculates the blend in the range 0-1. If nothing was hit, we set the blend to 1. Now, the environment is an infinite sphere surrounding the scene so it definitely qualifies for the far dist.

Our getBlend function is quite simple math. We subtract the nearDist from both the dist and the farDist, and then divide what's left of dist by what's left of farDist. Finally, limit it to the 0-1 range.

```
miScalar getBlend(miScalar dist, miScalar near, miScalar far)
{
        miScalar blend = dist - near;
        blend = blend / (far - near);
        blend = fmin(1, fmax(0, blend));

        return blend;
}
```

If you're on Windows, the fmin and fmax aren't included standard in your math.h, which is included. So, in this case, write your own version of each of these functions: fmin returns the smaller of two values passed, while fmax returns the larger. The implementation is very simple – you'll find it in the source code.

To get the amount of reflectivity, we mix between the near color and far color using our blend value. It's done exactly the same way you'd do it with paint. If you wanted halfway, you'd take half of the nearReflectivity and half of the farReflectivity and add them together.

```
miColor reflectivity = mixColors(nearReflectivity,
farReflectivity, blend);
```

And the mixColors function it calls.

```
miColor mixColors(miColor & a, miColor & b, miScalar & blend) {
        miColor mix;
        miScalar invBlend = 1 - blend;
        mix.r = (a.r * invBlend) + (b.r * blend);
        mix.g = (a.g * invBlend) + (b.g * blend);
        mix.b = (a.b * invBlend) + (b.b * blend);
        mix.a = (a.a * invBlend) + (b.a * blend);

        return mix;
}
```

The color our reflection ray returned was the color if it was 100% reflective. So, to get what our surface should actually reflect, we multiply the reflectivity we just calculated by the reflColor. **This code is even simpler, so just check it in image X** (in the downloaded resource files).

Finally to our reflectMode. If it's 0 (i.e. if (!reflectMode)), then we'll use mix rather than add. The only difference between the two is that we need to reduce the strength of the surface color. To do that, we'll multiply it by the inverse of the nearReflectivity. There's a good reason to use the nearReflectivity rather than the reflectivity we calculated. If we used the calculated one, then that would mean how reflective the actual surface is changes depending on how far away the reflecting object is. This isn't natural. So, by using the nearReflectivity, the surfaces reflectivity doesn't change – just the strength of the reflection.

Once that's done, we add the reflColor and the surface color, storing the result into out_pResult and finally returning miTRUE as our shader is done.

Hopefully this has explained a lot of the basics you'll need to come to grips with to understand coding some tools to enhance your rendering. If you're after a task to tackle, you could try expanding this shader. Features such as support for layering a specular on top (as at the moment the specular of a surface would also be dimmed in Mix mode) or glossy reflections are quite simple enhancements you could practice with.

Australian ***Alan Jones*** *started in the multimedia field, working on small bits of 3D for CD-ROM / Web development. He has been a SOFTIMAGE|XSI user since late 2001, when he started working with 3D full-time in the area of Architectural Visualization. He currently resides in London and works as a freelance SOFTIMAGE|XSI artist.*

INTRO TO NORMAL MAPPING

Intro to Normal Mapping

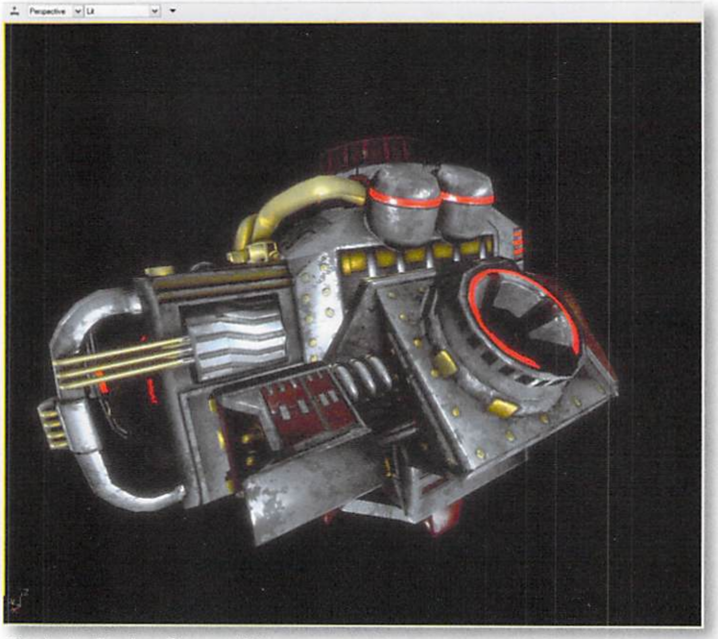


Fig 1– In Engine Screen #1

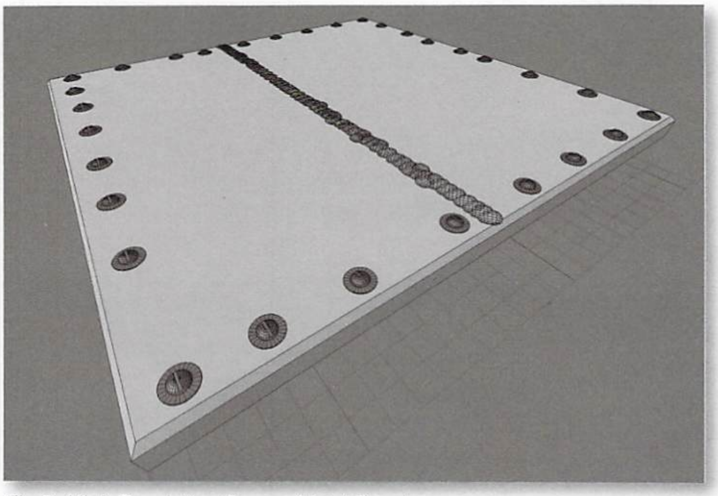


Fig 2- High Resolution Source Metal Panel

By Jake Carvey
3D Animator, Director, Producer
Los Angeles, California

The golden age of video game graphics is upon us. Several of the technologies and game titles shown at this year's E3 convention demonstrate a tremendous leap forward in visual quality. Real-time shadows, HDR rendering, and dozens of other innovations have given developers new tools for creating breathtaking interactive characters and environments. Welcome to the world of have your cake and eat it, too.

One of the most important leaps forward has been the implementation of "normal mapping" technology. Normal maps, also know as DOT3 bump-maps, utilize the latest developments in real-time 3D technology, leveraging recent developments in graphics hardware and software to deliver unprecedented detail and realism. Basically, normal maps are textures that interact with the lighting of a scene to make a simple mesh seem more geometrically complex. **(Fig. 1)**

The basic techniques for generating and using normal maps are fairly simple. However, successful implementation to take full advantage of all of the possible opportunities involves a great deal of aesthetic skill in various disciplines. Intimate knowledge of high-resolution and low-resolution modeling, photorealistic texturing, and near-flawless UV mapping techniques are all essential.

But don't let that scare you off. The bar has been raised in the game industry, and feature film skill sets are quickly becoming prerequisites for continued success as a real-time artist. Specializing in one area has been important in CGI film for a long time, and that will become more and more true in the gaming industry as well. However, it has never been more important to have a clear view of the big picture: how all of the specialized skills complement and tie into one another.

This article will demonstrate the basics of creating normal maps within a production pipeline for next-generation game engines. Knowledge of this technology is fast becoming a prerequisite skill for all artists working on projects for new game engines and platforms, such as the spectacular UnrealEngine3 on PC, Xbox360, and PS3 platforms.

MANUAL NORMAL MAP CREATION

To illustrate the basic concepts behind what is going on, we'll create a simple scene in any 3D package to extract a normal map from a complex, but relatively flat, object. This is a down and dirty (and free) method for creating normal maps that only works for relatively flat surfaces or surfaces that use tiled normal maps.

STEP 1:

Model a Hi-Res metal panel lying flat on the ground. Considering that most game engines still require textures with dimensions of "powers of 2" (256x256, 512x512, 1024x2048, etc), it's probably best to model your test panel as a square or a rectangle that has the same width to length ratio as your final texture.

STEP 2:

Add bevels, rivets, weld seams, or whatever other features you like. The trick is that these features should be fairly flush with the surface. Any features that will be raised above the surface any significant distance should be modeled into the Lo-Res model directly, and should be generated using the more sophisticated techniques discussed in the next section. You can also use bumps maps to add additional depth and character. **(Fig. 2)**

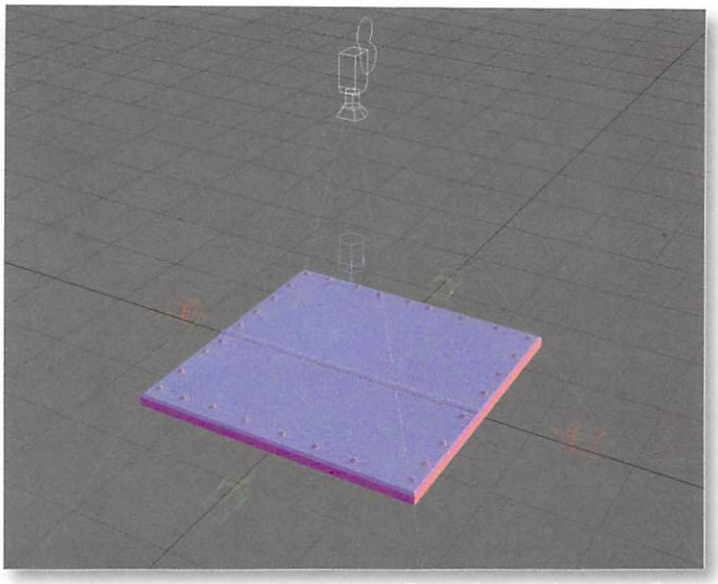


Fig 3- Camera & Lighting Setup

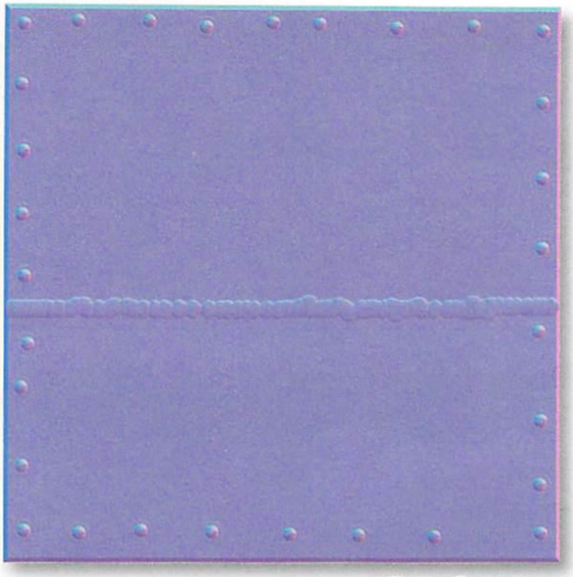


Fig 4- The Normal Map

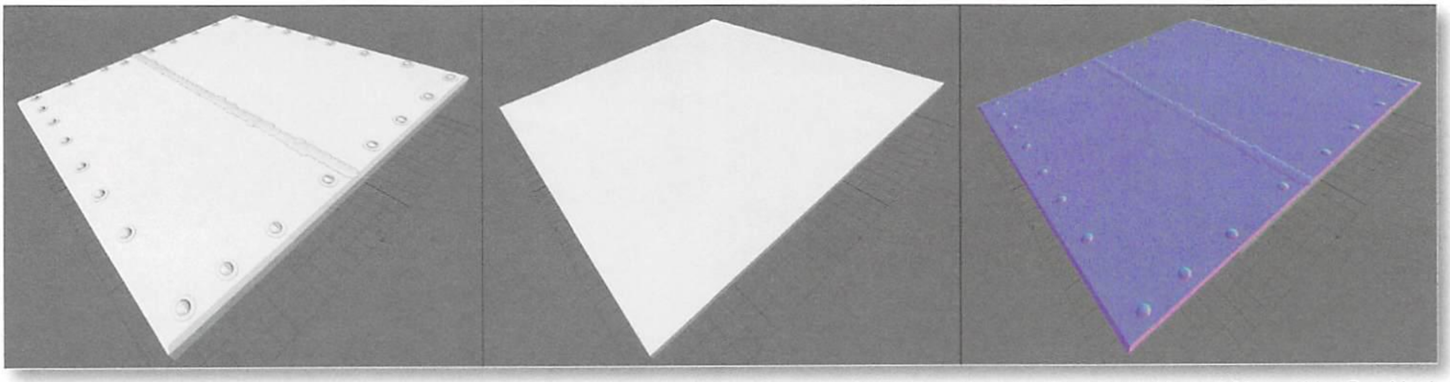


Fig 5- Hi-Res Mesh,Lo-Res Mesh, Normal-Mapped Lo-Res Mesh

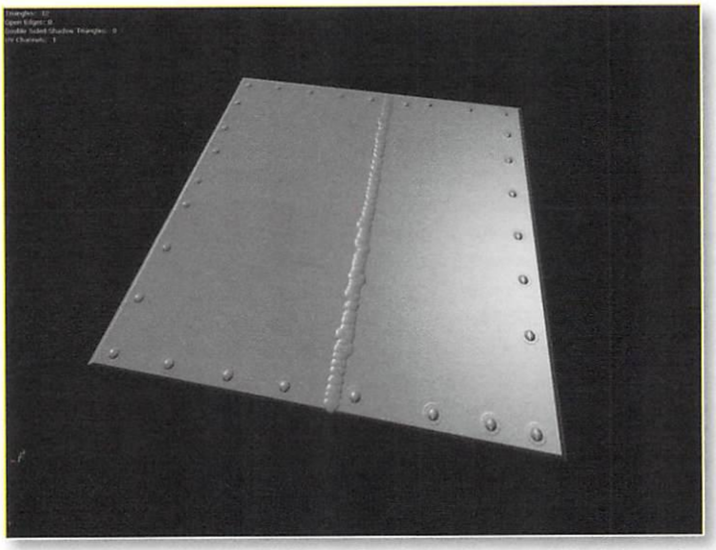


Fig 6- Final Normal Mapped Mesh in engine

STEP 3:

Adjust the surface/material properties so that the Diffuse color is pure white (255,255,255). Set all other Texture Channels to 0% (Black).

STEP 4:

Create five directional lights:

- Positive Blue (0,0,255) @ 50% intensity rotation: 0°,90°,0°
- Positive Red (255,0,0) @ 50% intensity rotation: -90°,0°,0°
- Negative Red (255,0,0) @ -50% intensity rotation: 90°,0°,0°
- Positive Green (0,255,0) @ 50% intensity rotation: 0°,90°,0°
- Negative Green (0,255,0) @ -50% intensity rotation: 0°,90°,0°

STEP 5:

Set the Ambient Lighting for the scene to 50% intensity (or set the Luminosity/Emissiveness of your Hi-Res surface to 50%)

STEP 6:

Create an orthogonal camera that frames your Hi-Res panel model tightly. If your 3D software doesn't have orthogonal cameras (LightWave, for instance), set a very long focal length and move the camera far away from the subject, until it completely fills the camera view. **(Fig. 3)**

STEP 7:

Render and enjoy! **(Figs. 4, 5, and 6)**

It is also possible to create normal maps on fairly flat real world objects using nothing more than a flashlight and a digital camera! *(See http://www.zarria.net for a detailed tutorial.)*

PAINTING NORMAL MAPS IN PHOTOSHOP

For certain types of work, it can be more precise to model a 3D source object, while in other situations, straight Photoshop skills are all that is

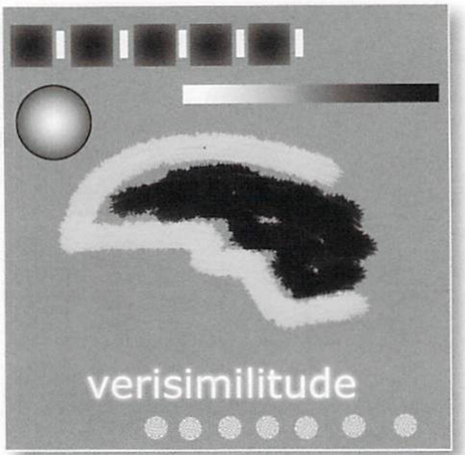


Fig 7– Height Map

necessary. You can make normal maps in Photoshop by painting a grayscale map in much the same way as you would create a bump or displacement map. To convert the grayscale image into a colored normal map, use the nVidia Normal Map filter, available for free at the nVidia developer Web site.

2D TEXTURE PIPELINE

STEP 1:

Create Lo-Res model, with solid UVs. Overlapping of identical parts is okay.

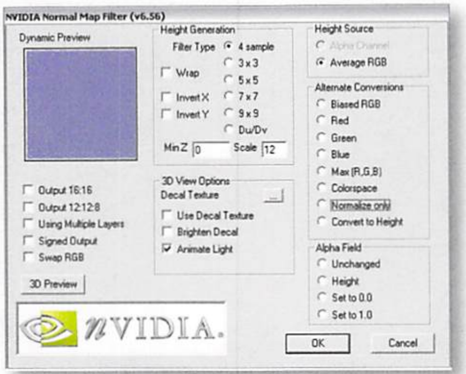
STEP 2:

Paint Lo-Res textures (for Color, Specularity, Emissive, etc.)

STEP 3:

Paint 2D grayscale displacement map. Use typical techniques for creating bump or displacement maps – white is high, black is low. **(Fig. 7)**

STEP 4:


NVIDIA Normal Map Filter (v6.56)

Dynamic Preview

☐ Output 16:16
☐ Output 12:12:8
☐ Using Multiple Layers
☐ Signed Output
☐ Swap RGB

Height Generation
Filter Type ● 4 sample ○ 3x3 ○ 5x5 ○ 7x7 ○ 9x9 ○ Du/Dv
☐ Wrap
☐ Invert X
☐ Invert Y
Min Z 0 Scale 12

3D View Options
Decal Texture
☐ Use Decal Texture
☐ Brighten Decal
☑ Animate Light

Height Source
○ Alpha Channel
● Average RGB

Alternate Conversions
○ Biased RGB
○ Red
○ Green
○ Blue
○ Max (R,G,B)
○ Colorspace
○ Normalize only
○ Convert to Height

Alpha Field
○ Unchanged
○ Height
○ Set to 0.0
○ Set to 1.0

Fig 8– nVidia Normal Map Filter

Convert displacement map to Normal Map via nVidia Normal Map plug-in for Photoshop. Use high values (12-48) in the options for a more pronounced effect. **(Figs. 8, 9, and 10)**

THE "IDEAL" 3D NORMAL MAP PIPELINE

Okay, so now that we have a basic understanding of what is going on, let's move on to the nitty-gritty of creating normal maps for full 3D shapes.

Here are the basic components of an ideal normal map production pipeline: **(Fig. 11)**

STEP 1:

Create a kick-ass high-resolution model (hereafter known as the "Hi-Res"). It is important for the designer and Hi-Res modeler to work closely to ensure parts and details that will effectively utilize the technology. **(Fig. 12)**

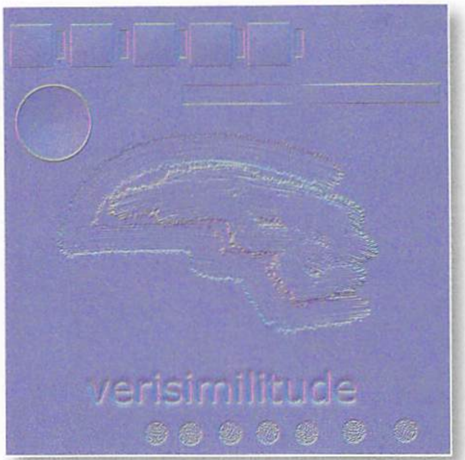


Fig 9– Resulting Normal Map

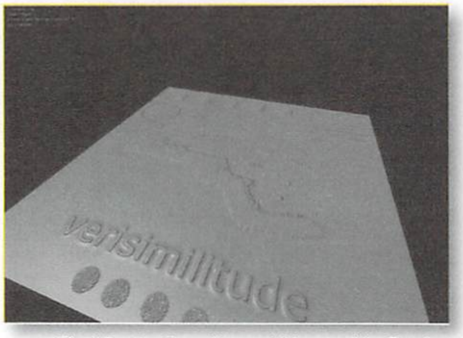


Fig 10– Lo Res Normal Mapped In Engine

STEP 2:

Create a kick-ass texture for the Hi-Res model. This texturing can take advantage of every texturing trick in the book, from procedurals, to UV mapping, to standard planar, cylindrical, cubic and spherical mapping, to decal mapping, and 3rd party shaders.

Reflection is handled during the

Fig. 11– Pipeline Pipe

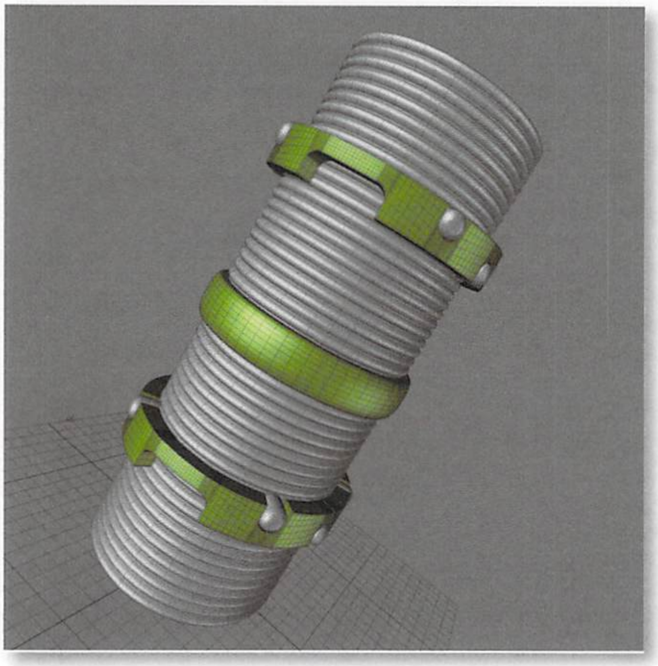
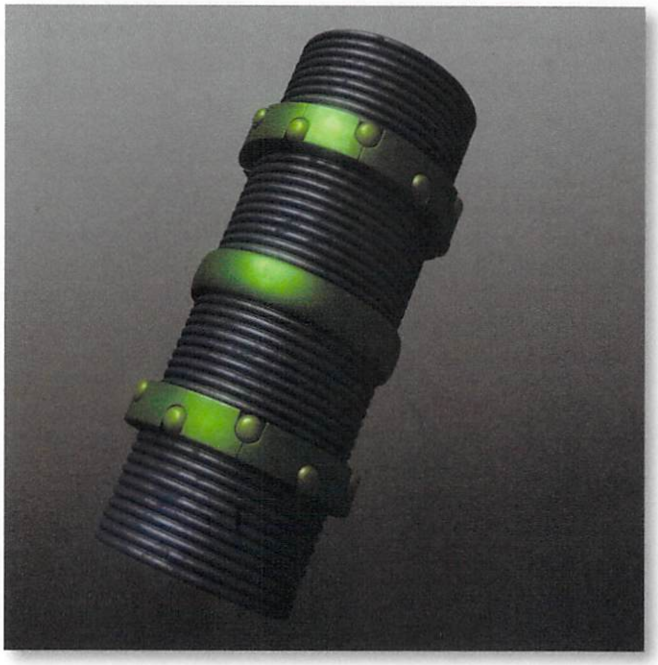


Fig. 12– Hi-Res Pipe



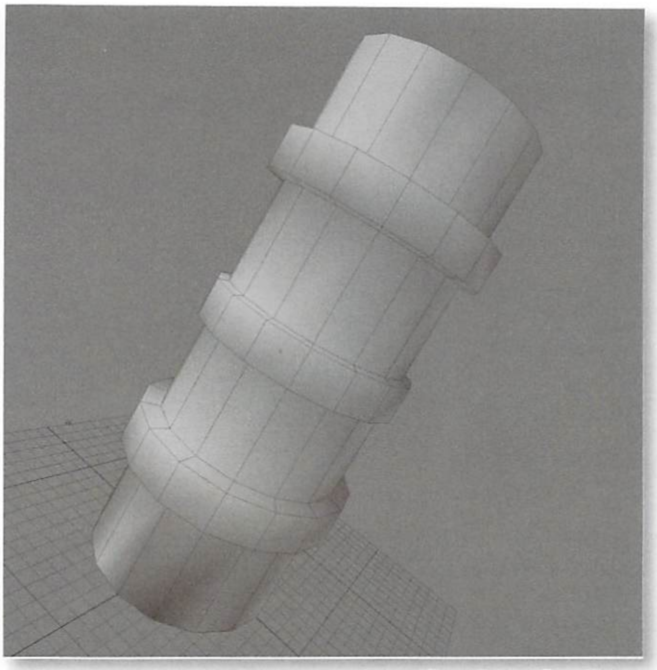


Fig. 13– Lo-Res Object

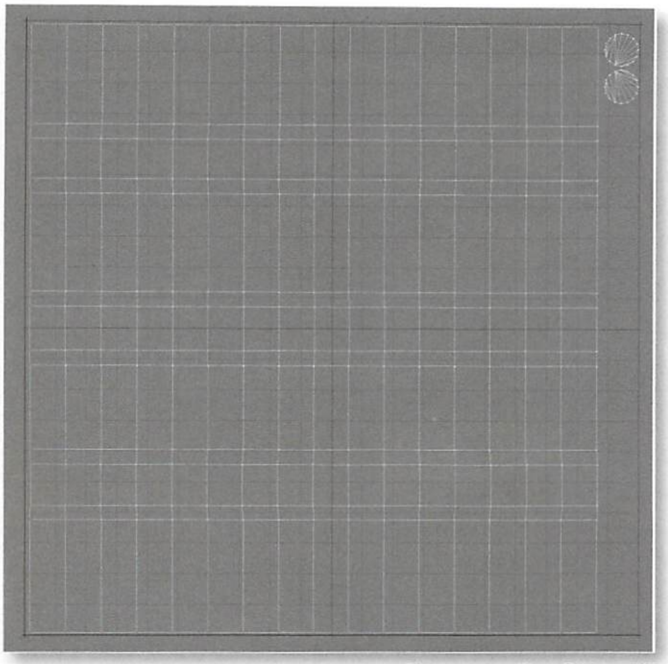


Fig. 14– Lo-Res Object UV

creation of real-time shaders, but there is no reason it can't be included during the texturing phase to approximate the final look of the model.

Everyone that sees the luscious renders of the fully textured Hi-Res model is going to ask: "Will it look like that in-game?" To which you reply: "Almost!"

STEP 3:

Create a game-ready low-resolution model (hereafter known as the "Lo-Res"). Topology should match Hi-Res as closely as possible, especially along prominent edges. The closer the Lo-Res model fits the shape of the Hi-Res source model, the better the normal map will turn out – especially along prominent edges. **(Fig. 13)**

Work within polygon budgets provided by project leads. For engines capable of utilizing normal map technology, acceptable poly counts are typically MUCH higher than they have been in the past. But be sure and leverage those extra polygons by making sure they end up in the right place.

Apply a single material to the Lo-Res object, and make sure you set the smoothing angles / groups to 180° - the normal map takes care of all of the bevels and sharp edges, etc.

STEP 4:

Create a super-clean, well laid-out UV Map. The accuracy and skill with which you lay out the UVs directly affects the flexibility and clarity of the final in-game model. Logically laid out UVs can mean the difference between being able to paint fixes directly into the texture map and having to run through the entire pipeline from the beginning. While a well-oiled pipeline can produce rapid results, being able to solve a problem with a texture revision is still quicker and poses far less risk that something else might break along the way. **(Fig. 14)**

STEP 5:

Bake out the necessary textures: Normal, Specularity, Ambient or Flat Color (with no shading), Diffuse Shading, etc. With most baking tools, the Normal, Color, Specularity, and other non-raytraced texture channels can render very quickly. If you need Ambient Occlusion or Radiosity passes, this can add quite a bit to the calculation time. I usually start with the basic passes, and then bake an Ambient Occlusion pass once everything is working just right. Then I usually multiply the Ambient Occlusion pass over the Color and Spec passes, which burns in the shadows and increases the overall contrast a bit.

The commercial plug-ins like Microwave and Kaldera can bake out all the necessary textures in a single pass, but most of the built-in or free plug-in solutions create normal maps through an entirely different interface than the tools used to bake out the other textures.

STEP 6:

Load the meshes and textures into the engine and watch jaws drop.

While we would all love to get everything right the first time, this technology evolves so quickly that significant asset changes are inevitable. So, a very important part of setting up a pipeline for these types of assets is that all of the steps are easily reproducible when you need changes. Something always needs more detail, or more rounded corners, or darker panel lines, to get the look just right.

I always suggest testing with the base textures first – if your UVs are laid out well, you can often paint more details and shading directly into the map. Also, if something is drastically wrong, it's better to know earlier than later.

As always, it's a trade-off between time, efficiency, and repeatability. Because, oh yes, the assets will ALWAYS need changes and tweaks (see below).

"THE WAY IT REALLY HAPPENS" PIPELINE:

STEP 1:

Create rough stand-in Lo-Res models so programmers and animators can start working and testing immediately.

STEP 2:

Follow Steps 1-6 in "IDEAL PIPELINE" above.

STEP 3:

Repeat Steps 1-6 in "IDEAL PIPELINE" as many times as necessary to make everything kick more ass.

Needless to say, having highly skilled technical directors and riggers will aid immensely in ensuring that no previous work is lost during the revision process, especially with characters or other models with complex animation.

BAKING

Maya has built-in tools for baking normal maps, and there are several additional

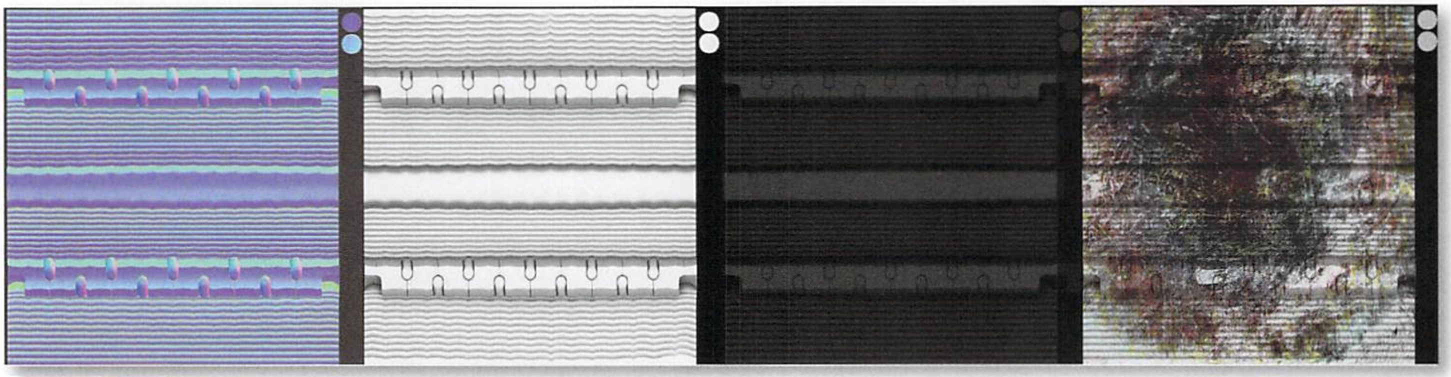


Fig. 15– Baked Texture

solutions available via HighEnd3D. Max 7 has built-in tools. There is a free plug-in for LightWave from Marvin Landis available via Flay.com. **(Fig. 15)**

Commercially available plug-ins add many additional features and ease of use, as well as increased speed and accuracy. Microwave3D, from Evasion3D is available for LightWave and Maya with mental ray standalone. Kaldera from Mankua appears to be an excellent 3rd party solution for Max.

nVidia and ATI offer free normal map generation tools, but they tend to be restrictive in the mesh formats they accept, and they don't bake other textures types, just normals.

TESTING:

Now, of course, none of this does you any good unless you can actually see the maps behaving as expected in real-time There are various plug-ins normal maps in your rendering application, but there's nothing quite as cool as seeing normal maps work in real-time as intended.

nVidia and ATI both have freely downloadable shader authoring applications which can be used for testing normal maps, as well as playing around with the latest in shader technology. They can be a bit overwhelming for non-technical types.

The Torque Engine from Garage Games supports normal maps, and is a fairly inexpensive game authoring environment. Quest3D is relatively inexpensive for the lite version and also supports normal mapping.

Several games are out now which use normal maps and include game editing tools: Far Cry, Chronicles of Riddick, and Doom 3 all use normal mapping and it is fairly simple to pick up the basics of the tools for testing your normal-mapped models. The game "modding" communities for these tools are very active, and there are dozens of forums dedicated to these particular games. These games still use fairly simple material implementations – but keep an eye out for the release of the next episode in the Unreal Tournament series. If that game ships with an editor, you will truly have access to some awesome tools!

TIPS AND TRICKS
MAXIMIZING THE EFFECT

- Use geometry to add overlapping and protruding elements as often as possible. These will cast true shadows and help a great deal in "selling" the look of the normal mapped surface
- Create seam-lines and part-lines around all important features on the Hi-Res. Never rely on the normal map alone to create clearly shadowed seams. Enhance seams and distinct depressions with darkened surfaces in the High-Res model.
- You can also blend an Ambient Occlusion pass into the color map to enhance the in-engine shading and shadowing. In fact, this is a very important step in almost all cases.

SMOOTHING ANGLES

- When baking normals from a Hi-Res source, set the smoothing angles on your Lo-Res object to 180 degrees (full smooth). The normal map will handle shading the sharp corners and bevels correctly. When using normal maps painted in Photoshop, extensive testing is necessary to manually set appropriate smoothing angles for the best result.

UV MAPPING

- The cleaner the UVs the better the results. Spend the time to get these right. It is fine to overlap UVs of identical parts, although we find that you can get a clear baked result, if you delete or disable the duplicate parts until after a successful baking session.
- For the highest quality Normal Mapping, I find that "Automatic" UV mapping tools can create a lot of problems. While very accurate, they can introduce seams and other artifacts into the final texture. Whenever possible, ensure that important, high visibility edges are connected – especially for areas that need clean Normal-Mapped bevels and other details across a single edge.

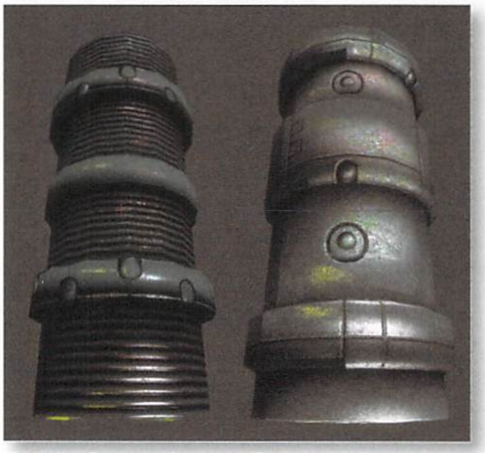


Fig. 16– TexturedPipes In Engine

- Baking different parts of an object separately is often very useful too, even if it will eventually use the same UV map and texture. If you lay out the UVs properly, you can quickly and easily re-combine the individual images in Photoshop.

RESOURCES:

http://developer.nvidia.com
http://www.ati.com/developer/

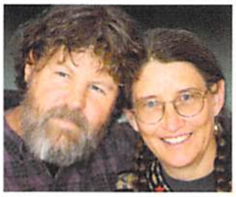
http://luxology.com
http://alias.com
http://newtek.com

http://evasion3d.com
http://www.mankua.com

http://highend3d.com
http://flay.com
http://www.zarria.net
http://www.monitorstudios.com/bcloward

Jake Carvey *has been directing, producing and creating 3D animation for broadcast, games and film for over 10 years. He has worked with clients such as Sony, Disney, Universal Studios, Dunkin' Donuts, ABC, and Kid's WB!, winning numerous industry awards along the way. He loves his work because the hours are great.*

LightWave Essentials: *Lscript Commander*



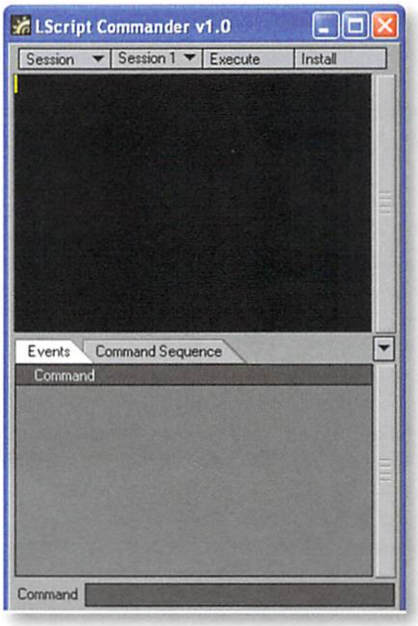
By Brad Carvey
Electrical Engineer,
3D Animator
New Mexico

Lscript Commander is a plug-in that makes it easy to create your own custom plug-ins. The Lscript Commander records most LightWave activities as commands and displays them in a command window. Selecting and dragging commands from the Lscript Commander command window, on the bottom, to the session window on the top allows you to modify, delete, or execute them. After recording a sequence of commands and adding them to the session window, you can execute the session or create a plug-in. It's easy to use Lscript Commander to automate redundant tasks.

In this tutorial, we will create a 3-point lighting setup plug-in. Lscript Commander records each step, and then creates a plug-in. The new plug-in will be assigned to an Lscript Macro button. Every time a scene is loaded, a click of the Lscript Macro button automatically adds a 3-point lighting setup.

During the tutorial, you may make some mistakes that record as commands; you cannot edit them. Start over, if needed, by clicking "Clear Scene" (N) and removing the Lscript Commander from the Master Plug-ins. Or, after you drag all of the commands to the session window, you can select a command and hit the delete key to remove it.

Figure 1



START LSCRIPT COMMANDER

1. Open Layout.
2. Open Master Plug-ins (Ctrl Q). With 7.5 use Layout/Plug-ins/Master Plug-ins to open.
3. Add Lscript Commander.
4. Double-click Lscript Commander to open the Lscript Commander (**not** "Lscript"). SEE **FIGURE 1**.
5. Close Master Plug-ins window; leave the Lscript Commander window open.

3-POINT LIGHTING SETUP

6. Select Light (Shift L). EditLights adds to the Lscript Commander Command window.
7. Change Light Name.

LW 8.0

a Click Light Properties (p). This will open the Item Properties window for the light, and add "ItemProperties" to the command window. (With each step more commands will be added. When commands are selected and dragged to the upper session window, they can be executed.)

b Highlight the Current Light Name and change it to "Key_Light" and press Enter key. *Note:* It is important that there are no spaces in the name, e.g. Fill_Light, or FillLight, **not** Fill Light.

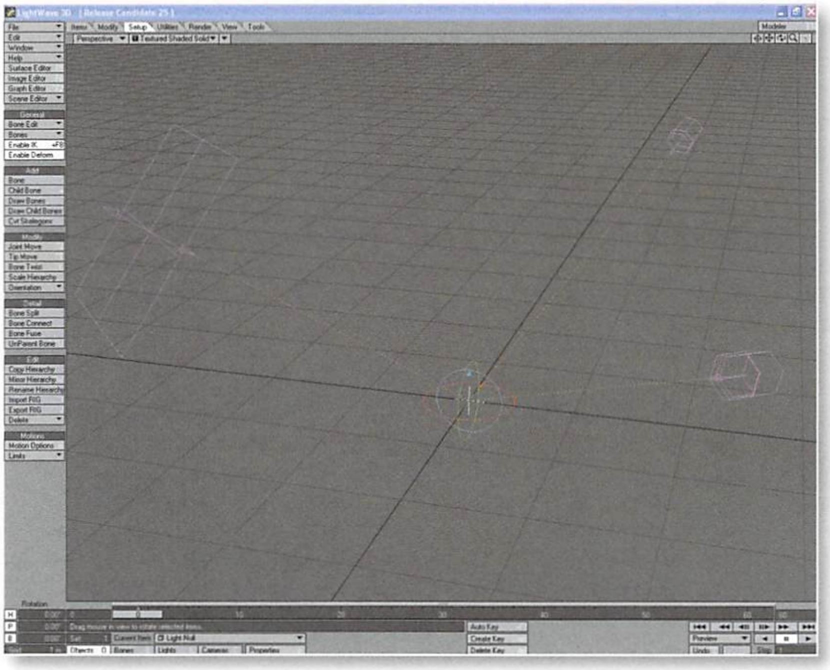
LW 7.5 Open the Scene Editor, right click "Light," and rename to "Key_Light."

8. Add a Distant Light and name it "Fill_Light."
9. Add a Distant Light and name it "Back_Light."
10. Add a Null (ctrl n) and name it "Light_Null."
11. Select Light Selection Mode (Shift L).
12. Select Motion Options (m).
13. Select Each Light and change its Parent Item and Target Item to "LightNull." **SEE FIGURE 2**.
14. Click Light Properties (p).
15. Select Key_Light as the current light and change the Light Intensity to 65%. Change the Light Type to "Area Light."
16. Select "Fill_Light" as the current light and change its Light intensity to 30%.
17. Select "Back_Light" as the current light and change its Light intensity to 40%. Close the Light Properties.
18. Select the "Fill_Light," move it to up (+y) and to the right (+x), then create a keyframe.
19. Select The "Back_Light," movie it up (+y), back (+z), and to the right (+x), then create a keyframe. **SEE FIGURE 3**.

CREATE 3-POINT LIGHTING PLUG-IN

20. In the Lscript Commander Events command list, scroll up and select "EditLights."
21. Scroll to the end of the commands, hold the Shift Key down and select

Figure 2



the last command. While keeping the left mouse button pressed drag the mouse pointer to the Black area above the Event Commands and release the button. The 3-Point Lighting setup commands should appear in the upper window. SEE **FIGURE 4**.

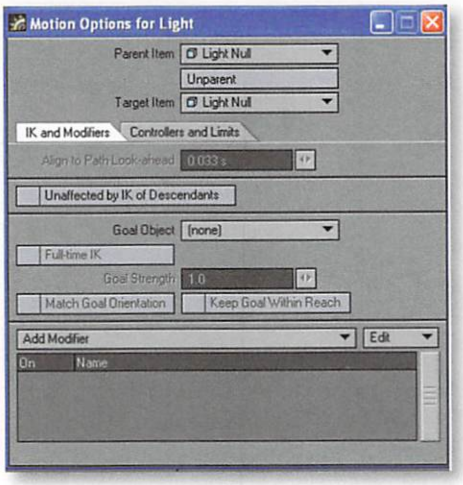
22 Click the Install button at the top right of the Lscript Commander.

23 Enter "3 lights" in the "Name Script" field and click OK. This creates a Macro Tab if one does not already exist and adds the "3 Lights" button.

A Macro Tab will be created in the "Lscript" Menu, if one does not already exist, and the "3 Lights" button will be added.

To use the new Lscript, start by opening a new scene or clearing the scene (N). Clicking the "3 Lights" button adds the lights. You can adjust the lights by moving and rotating the "Light Null." To have a little more control over lighting angles, remove "Light Null" as a target and adjust the position and angle of that light. It is also possible to create a variety of lighting Lscripts and add them to a new tab called "Lighting."

As an introduction to Lscript and Lscript Commander, I chose an easy but useful example. Use Lscript Commander to simplify lots of repetitive tasks. Lee Stranahan has some excellent free Lscript tutorials on NewTek's Web site:

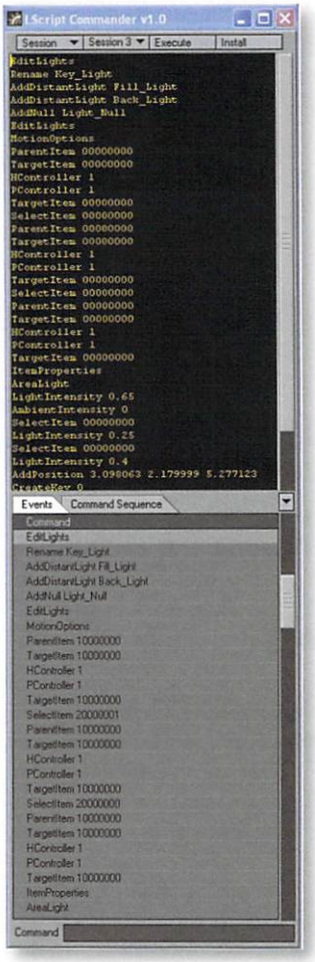


Parent Item: Light Null
Unparent
Target Item: Light Null
Align to Path Look-ahead: 0.033 s
☐ Unaffected by IK of Descendants
Goal Object: (none)
Goal Strength: 1.0
☐ Match Goal Orientation
☐ Keep Goal Within Reach

Figure 3

http://www.newtek.com/products/lightwave/tutorials/videos/index.html

Brad Carvey *has been doing computer animations for a long time. In 1969 brad used an analog computer, which was the size of a car, to produce his first computer animation. Brad is an electrical engineer and an emmy award-winning member of the Video Toaster development team. He prefers to do feature film work. his credits include films like* ***Men in Black, Stuart Little, Black Hawk Down, Kate & Leopold****, and* ***Master of Disguise****.*



```
EditLights
Rename Key_Light
AddDistantLight Fill_Light
AddDistantLight Back_Light
AddNull Light_Null
EditLights
MotionOptions
ParentItem 00000000
TargetItem 00000000
HController 1
PController 1
TargetItem 00000000
SelectItem 00000000
ParentItem 00000000
TargetItem 00000000
HController 1
PController 1
TargetItem 00000000
SelectItem 00000000
ParentItem 00000000
TargetItem 00000000
HController 1
PController 1
TargetItem 00000000
ItemProperties
AreaLight
LightIntensity 0.65
AmbientIntensity 0
SelectItem 00000000
LightIntensity 0.25
SelectItem 00000000
LightIntensity 0.4
AddPosition 3.098063 2.179999 5.277123
CreateKey 0
```

```
EditLights
Rename Key_Light
AddDistantLight Fill_Light
AddDistantLight Back_Light
AddNull Light_Null
EditLights
MotionOptions
ParentItem 1000000
TargetItem 1000000
HController 1
PController 1
TargetItem 1000000
SelectItem 2000001
ParentItem 1000000
TargetItem 1000000
HController 1
PController 1
TargetItem 1000000
SelectItem 2000000
ParentItem 1000000
TargetItem 1000000
HController 1
PController 1
TargetItem 1000000
ItemProperties
AreaLight
```

Figure 4

XSI base.com
Softimage XSI Community

Forums
Jobs
Interviews
Plugins
Shaders
Contests

www.xsibase.com

COMPUTER & COMMUNICATION CENTRE
STD ISD PCO
ail,Hot-mail

Time to upgrade?

Welcome to Smedge 3, the next generation in distributed processing.

Rebuilt from the ground up, the next generation of Smedge provides unbeatable power and flexibility combined with incredible stability. All while maintaining the ease of use artists have always loved about Smedge. And combining the power of your Windows, Linux and Mac machines will soon be a snap with the amazing cross-platform abilities.

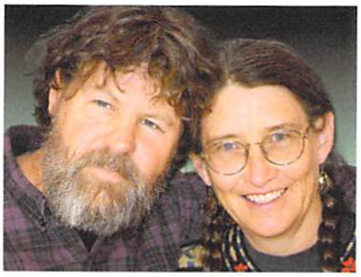
For more information, please visit:

http://www.uberware.net/smedge3/

Überware

FINAL RENDER:

POOPING IN THE RING



By Andrea Carvey
Archeologist, Animator
New Mexico

I trust that you are familiar with the phrase "harder than herding cats." I also assume that you are familiar with the TV show that showcases funny animal videos, particularly commercial, television, or movie bloopers, in which animals behave ... shall we say ... unexpectedly. This is understandable for some of the less intelligent creatures, but even highly trained animals have been known to confound their owners at the most inopportune times. Talk shows regularly invite animal "experts" and their animals to the program for a grownup version of Show & Tell. Usually, they bring a number of unruly beasts onto the stage, resulting in chaos and high jinks. Great television!

Things are a little different in the show ring. In the ring, animals are supposed to behave themselves. Now, I'm not sure about the rules for showing dumber animals, such as rabbits or guinea pigs, but one of the things that can get you disqualified when showing dogs or horses is if your animal poops in the show ring. Think about it. Have you ever seen a dog in the Westminster Dog Show taking a crap in front of the judge? Or a horse dropping some turds as it flies over a fence at the Olympics? I don't think so. Real pros, those animals.

I wish I could say the same about computers. Everyone always says how smart they are and all, but I know differently. (You ***know*** I'm right.) I have yet to experience a computer that has *not* become so totally confused that it just froze up, exactly like a guinea pig that I once owned that would get lost in a corner ... back and forth, from one side of the corner to the other, never finding its way out. And to make matters worse, it always happens at just the wrong time!

A couple of months ago, Brad had an important presentation to give. It was a press conference with some politicians and other rather impressive-sounding people. Now, Brad is really good at giving these kinds of presentations so that everyone is entertained, as well as educated, but it gets harder for him if he has to work a computer at the same time, because (as we all know) working with computers is a lot like working with animals. So, I suggested that I work the computer so he would be free to just talk. Great!

Everything was set up and working. I was familiar with the program and had been using it problem-free for months – no crashing, no weird behavior. Perfect. Soon, people began coming in and mingling, the TV cameras were set up, and I was informally showing a few people what I was doing ... again, no problem. Then, it was announced that the presentation was about to begin, so I started to reset the program so I'd be all ready. I swear, I wasn't nervous or anything. It was all very routine. But low and behold, at just that moment, guess what? The computer ... pooped in the ring!

What the heck?

Oblivious to the disaster that was taking place on the screen behind his back, Brad began his presentation.

Uh, let's see. Maybe I did something wrong.

I tried again.

Poop.

Hmmm. Uh, maybe if I do this?

Poop.

What about this?

Poop.

What the heck is going on?!

OK, OK, I'll go back and reboot.

(Mind you, the television cameras are getting *all* of this, and Brad is continuing on, still thankfully unaware of the situation.)

Poop.

Poop. Poo-oo-oop!

Ahhhhhhg!

What I had on my hands was not merely a case of pooping in the ring. Pooping in the ring happens and then it's over. This was diarrhea! Uncontrollable, explosive, ***public,*** diarrhea!

What does one do in this situation? If this were a dog show, my dog would be disqualified and I would have to take it out of the ring. No such mercy here. No, I couldn't just hang my head in shame and gratefully be allowed to vanish. The show must go on! It was clear that I was not going to get anywhere. I just had to make the most of it, so I brought up a different program, similar to the intended one, and fiddled with it for a while, trying to pretend that it had *anything* to do with what Brad was saying.

At that point, Brad suggested that I demonstrate some particular feature. Reluctantly, I had to tell him (calmly) that I've been having a *little* trouble. No need to freak him out now. Considering it all, he took it quite well and was able to go on to something else.

Mercifully, it was all over quickly. No one mentioned the turd in the ring. I tell myself that probably only a few computer-savvy people actually realized what was going on, and they (having apparently made some sort of divine bargain of their own) were keeping their pledge to ignore such calamities. I like to think that's true.

Of course, this particular problem has never occurred again. So much for trying to figure out what went wrong. I guess the computer got it all out of its system ... all at once ... in the ring ... until the next time.

Brad and Andrea Carvey** have been doing computer animations for a long time. In 1969 brad used an analog computer, which was the size of a car, to produce his first computer animation. Andrea, an archeologist, prefers to do scientific animations; her credits include programs like Discovery Channel's **Understanding Cars.** Brad is an electrical engineer and an emmy award-winning member of the Video Toaster development team. He prefers to do feature film work. his credits include films like **Men in Black, Stuart Little, Black Hawk Down, Kate & Leopold,** and **Master of Disguise.

FINANCING AVAILABLE!

TSUNAMI

EDIT. ANIMATE. FAST.

TSUNAMI LightWave™ 3D Workstations

We have taken the most complete and flexible software solution for 3D graphics and animation and coupled it with our dual Xeon® TSUNAMI Force to create the best solution available for the professional animator. Proven for years in television, film, and games, LightWave 3D [8] is versatile enough to make the transition to and from all kinds of projects. It is a full, robust program that includes many of the tools that other packages require to be purchased separately: soft-body dynamics, particles, hair and fur, unlimited render nodes, and more. The ultra-powerful TSUNAMI can be custom-built to your ideal specifications from a long list of components. Price yours today at www.sharbor.com/build-yours

NEW!

Cinema 4D XL LightWave Companion Bundle

Attention LightWave 3D users! Take advantage of this bundle that includes Cinema 4D XL R9, a special interface that looks like LightWave 3D, tutorials for LightWave users, and Sniper Pro, a plug-in that allows fast preview renders 899.99

NEW!

StoryBoard Artist 4

No more wasted time - plan your project in advance! Create professional quality digital storyboards with StoryBoard Artist. This easy to learn but sophisticated tool allows you to communicate your vision 749.00

StoryBoard Quick 4 . . . 199.00

NVIDIA Quadro FX1300 by PNY

This award-winning, ultra-stable 128MB PCI Express graphics board is designed to provide users the highest performance for 3D animation, compositing, motion graphic, and video editing applications 619.00

FX540 Pro Vid Edition . 274.99

Now use on a PC!

30" Cinema Display with Parhelia DL256MB PCI

Now PC users can enjoy the Apple Cinema Display! The Parhelia DL256 PCI graphics card is designed to drive one dual link DVI display, plus an additional analog display or video monitor 3549.99

Parhelia Card Only . . . 679.99

NEW!

Swift 3D LW v3

This powerful rendering plug-in for LightWave 3D enables you to render and export 3D scenes and animations into web-optimized vector or raster Macromedia Flash animations, as well as Adobe Illustrator, EPS, SVG, and PNT for U.S. Animation 269.00

Signature Courseware

Learn LightWave 3D from expert animator and author Dan Ablan, with the ultimate training for LightWave 3D. This 18-hour course on DVD-ROM will have you understanding modeling, texturing, lighting, animation, and rendering faster than you thought possible 269.00

WaveGuide Tutorials

Four popular tutorial CDs from Pat Beck that will get you animating with LightWave 3D [8]! Topics covered include compositing, dynamics, 2D animation, and a beginner's guide for non-animators. Choose any title for 40.99

All four titles 155.99

SoftImage|XSI Foundation

Get everything you need to take your work to the next level of realism: Two-CPU mental ray® v.3.3 rendering, advanced subdivision surface modeling, full character rigs, Cloth & Particles, and more! Training DVDs included 585.99

Competitive Upgrade . 409.99

Terms: POs accepted from schools and government agencies • All checks require 5-7 days to clear • Defective products replaced promptly • RMA number required for all returns • Returns accepted within 15 days, in original packaging, postage prepaid, undamaged • Opened software is not returnable • Shipping charges are not refundable • Returns are subject to a 15% restocking fee • Not responsible for typos • Prices are subject to change

NewTek™ Authorized Reseller

Authorized Reseller

Information
262-548-8120

Tech Support/RMAs
262-548-8157

Mon-Fri 8:30am-5:00pm CST

Fax: 262-548-8130

Safe Harbor Computers
W226 N900 Eastmound Dr.
Waukesha, WI 53186

www.sharbor.com

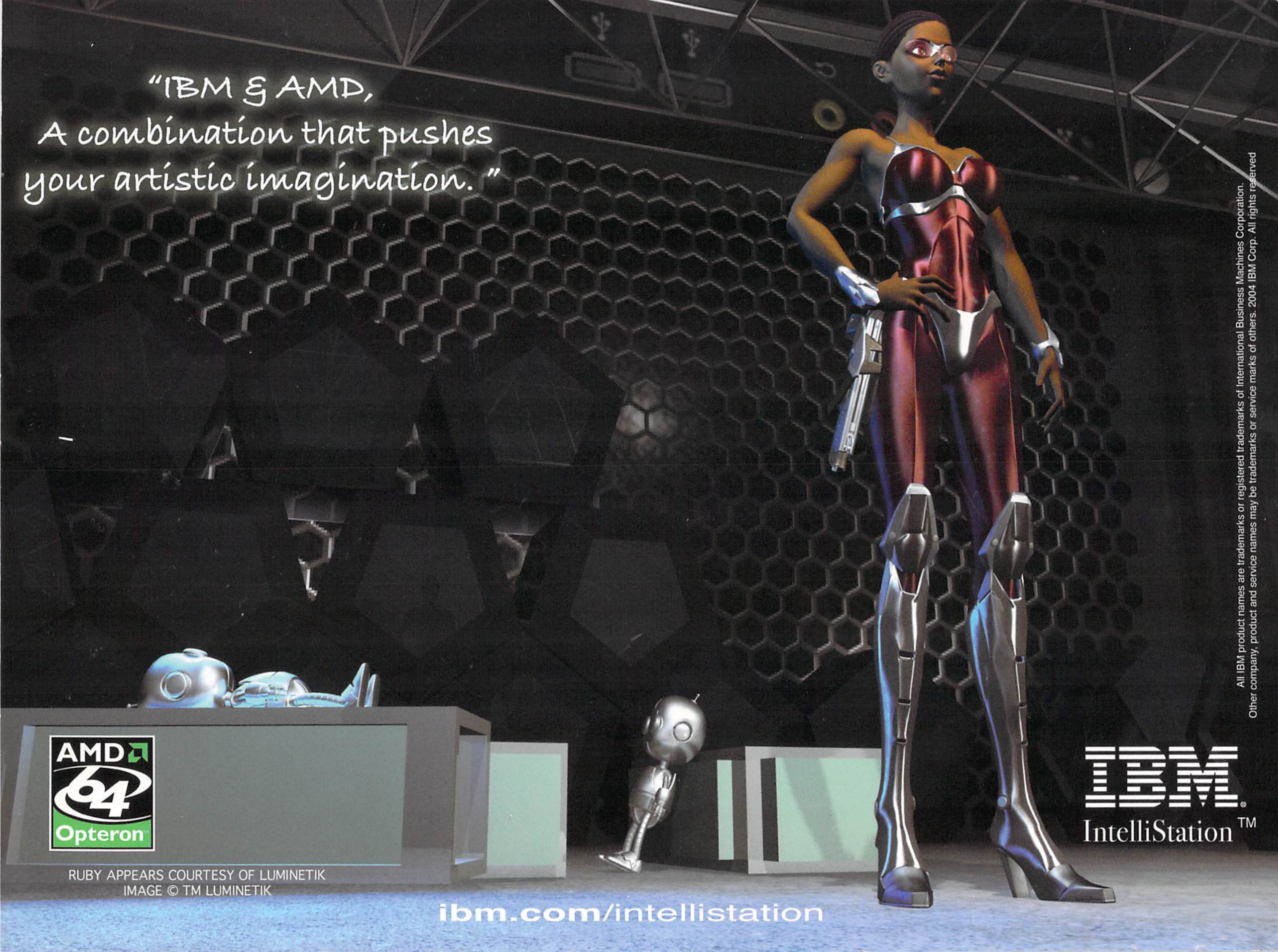
© 2005 Safe Harbor Computers. All rights reserved. TSUNAMI and the TSUNAMI logo are trademarks of Safe Harbor Computers.

YOUR SOURCE FOR ALL OF YOUR ANIMATION NEEDS!

www.sharbor.com

800-544-6599

SAFE HARBOR COMPUTERS


"IBM & AMD,
A combination that pushes
your artistic imagination."
AMD
64
Opteron
RUBY APPEARS COURTESY OF LUMINETIK
IMAGE © TM LUMINETIK
ibm.com/intellistation
IBM
IntelliStation ™

All IBM product names are trademarks or registered trademarks of International Business Machines Corporation.
Other company, product and service names may be trademarks or service marks of others. 2004 IBM Corp. All rights reserved